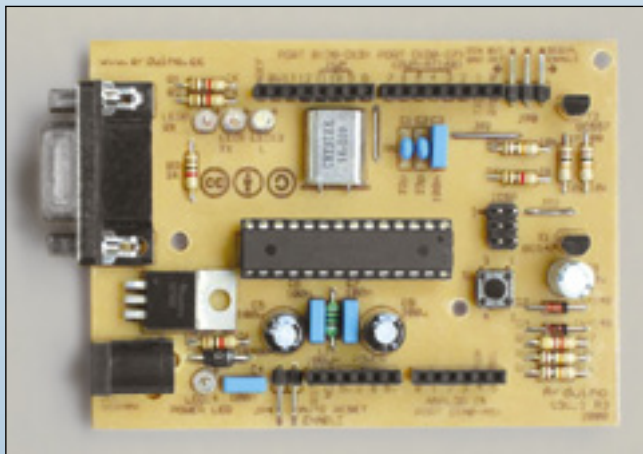
Conoscere e usare Arduino

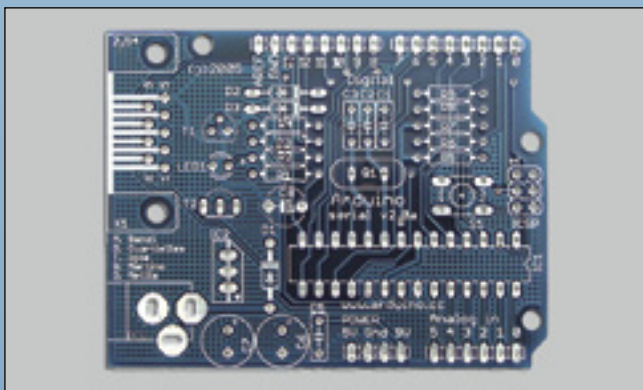
dell'ing. MIRCO SEGATELLO

È tra i sistemi di sviluppo più noti e utilizzati: si basa su un processore Atmel e dispone di numerosi I/O. Vediamo come funziona e come si utilizza. Prima puntata.

Arduino è il nome di una piattaforma hardware per lo sviluppo di applicazioni basate sui microcontrollori ATMEL. Ideata in Italia nel 2005, è basata su una semplicissima scheda di I/O e su un ambiente di sviluppo che usa una libreria Wiring per semplificare la scrittura di programmi in C e C++ da far girare sulla scheda. Wiring è un ambiente di programmazione open-source per impieghi su schede elettroniche, pensato per una facile applicazione; si tratta di un progetto italiano nato ad Ivrea (da un team composto da Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis) e successivamente sviluppato all'università Los Andes in Colombia. Arduino può essere utilizzato per lo sviluppo di oggetti interattivi stand-alone, ma può anche interagire con software residenti su computer, come Adobe Flash, Processing, Max/MSP, Pure Data, SuperCollider. La piattaforma hardware Arduino è distribuita agli hobbisti sia attraverso Internet che tramite fornitori locali ed è disponibile in versione pre-assemblata, mentre le informazioni sul progetto hardware (nel pieno rispetto della filosofia open-source) sono rese disponibili a tutti, in modo che, chiunque lo desideri, può costruirsi un clone di Arduino con le proprie mani. Il progetto Arduino ha preso avvio in Italia ad Ivrea, nel 2005, con lo scopo di rendere disponibile a progettisti, studenti e semplici hobbisti, un dispositivo di svilup-

Arduino Single-Sided Serial.

po facile ed allo stesso tempo più economico rispetto a simili sistemi di prototipazione. I progettisti sono riusciti nell'intento di creare una piattaforma di semplice utilizzo ma che,

Il modulo Arduino serial.

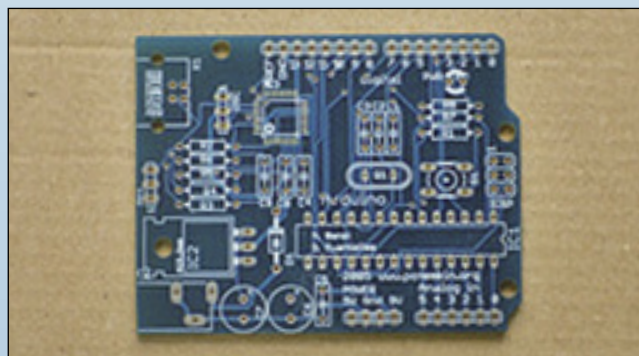
al tempo stesso, permettesse una significativa riduzione dei costi rispetto a molti prodotti disponibili sul mercato. A ottobre 2008 erano già stati venduti più di 50.000 esemplari di Arduino in tutto il mondo.

Circuito stampato dell'Arduino serial V2.0.

Una scheda Arduino consiste di un microcontroller a 8-bit AVR prodotto dalla Atmel, con l'aggiunta di componenti complementari che ne facilitano l'utilizzo con altri circuiti.
Le schede ufficiali usano i chip della serie megaAVR (nello specifico, i modelli ATmega8, ATmega168, ATmega328, e ATmega1280) ma i loro cloni sono disponibili anche con altri microcontrollori. Molte schede includono un regolatore lineare di tensione a 5 volt ed un oscillatore a quarzo da 16 MHz (o, in alcune varianti, un risuonatore ceramico), sebbene alcuni casi, come ad esempio LilyPad, girino ad 8 MHz e facciano a meno dello stabilizzatore di tensione.
La scheda Arduino è pre-programmata con un bootloader che semplifica il caricamento dei programmi nella memoria Flash incorporata nel chip, rispetto ad altri dispositivi che richiedono, solitamente, un programmatore esterno.
A livello concettuale, tutte le schede vengono programmate attraverso una porta seriale RS-232, ma il modo in cui questa funzionalità è implementata nell'hardware varia da versione a versione. Le schede seriali Arduino contengono un semplice circuito traslatore di livelli che permette la conversione tra il livello della RS-232 e quello dei segnali TTL.
Le recenti versioni di Arduino (Diecimila e Duemilanove) vengono gestite via USB, grazie a un'implementazione che usa un chip adattatore USB-seriale come l'FT232 della FTDI. Alcune varianti, come la Arduino Mini e la versione non ufficiale Boarduino, usano una scheda o un cavo adattatore USB-to-serial separato.
Le schede Arduino dispongono di molti connettori di Input/Output usabili quale estensione per altri circuiti esterni. La Diecimila, ad esempio, offre 14 connettori per l'I/O digitale, sei dei quali possono produrre segnali PWM; esistono poi sei ingressi per segnali analogici. Questi pin sono disponibili sulla parte superiore della scheda, mediante connettori femmina a passo 0,1 pollici.
Inoltre, sono disponibili commercialmente molte schede applicative plug-in, note come "shields".
Le schede Barebones e Boarduino e Seeduino, tre cloni compatibili con la Arduino, sono dotate di connettori maschio sul lato inferiore

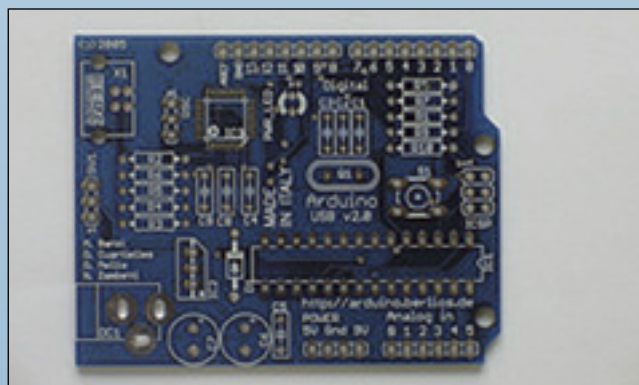
del circuito, in modo da poter essere connesse a una breadboard senza necessità di effettuare saldature.
L'ambiente di programmazione integrato (IDE) di Arduino è un'applicazione multipiattaforma scritta in Java, ed è derivata dall'IDE creato per il linguaggio di programmazione Processing e adattato al progetto Wiring. È concepito per introdurre alla programmazione hobbisiti e neofiti, a digiuno di pratica nello sviluppo di software.
Per consentire la stesura del codice sorgente, il programma include un editor di testo dotato di alcune particolarità, come il syntax highlighting, il controllo delle parentesi e l'identificazione automatica delle istruzioni. L'editor è inoltre in grado di compilare e lanciare il programma eseguibile in una sola passata e con un singolo click. In genere non c'è bisogno di creare dei Makefile o far girare programmi dalla riga di comando.
L'ambiente di sviluppo integrato di Arduino è fornito di una libreria software C/C++ chiamata "Wiring" (dall'omonimo progetto Wiring) che rende molto più semplice implementare via software le comuni operazioni input/output. I programmi di Arduino sono scritti in C/C++, ma, per poter creare un file eseguibile, all'utilizzatore non è chiesto altro se non definire due funzioni:

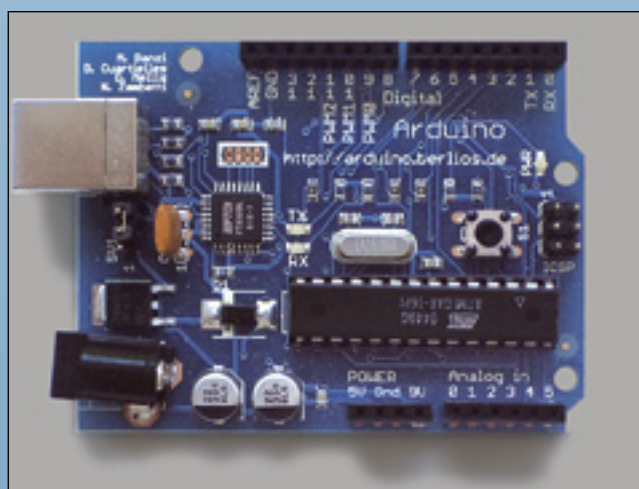
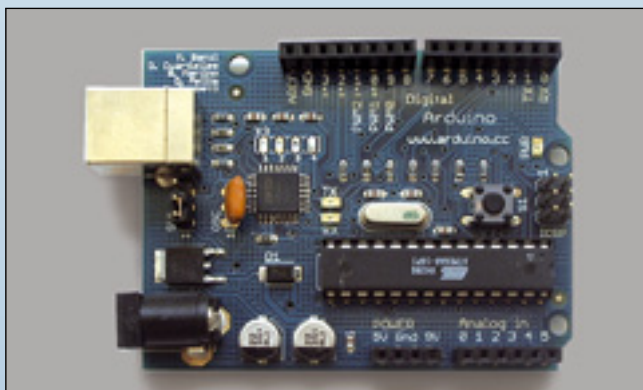
- setup() ; è una funzione invocata una sola volta all'inizio di un programma che può essere utilizzata per i settaggi iniziali;
- loop() ; è una funzione chiamata ripetutamente fino a che la scheda non viene spenta.

L'IDE di Arduino usa la GNU toolchain e la AVR Libc per compilare i programmi, mentre si avvale di *avrdude* per caricarli sulla scheda.
L'hardware originale Arduino è realizzato dalla italiana Smart Projects ed alcune schede a marchio Arduino sono state progettate dalla statunitense SparkFun Electronics.
Fino a oggi sono state commercializzate le seguenti versioni dell'hardware Arduino.

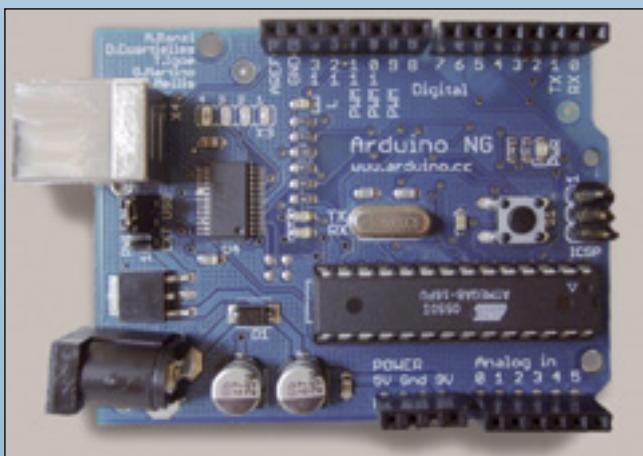
Arduino Single-Sided Serial: la primissima versione, equipaggiata con un ATMega8 e programmabile via seriale. Lo stampato è di tipo a singola faccia con tutti componenti DIP ed è quindi facilmente realizzabile a livello hobbistico.

Circuito stampato dell'Arduino USB.

Arduino serial: versione costruttivamente migliorata con PCB professionale, programmabile via seriale con microcontrollore ATMega8.

Arduino serial V2.0: versione migliorata della Arduino serial.

Stampato dell'Arduino USB V2.0.

Arduino USB: versione con connessione USB facente uso del convertitore FT232BM. La programmazione avviene connettendola via USB ad un PC.

Modulo Arduino extreme.

Arduino extreme V2.

Arduino USB V2.0: è la seconda versione della Arduino USB. In essa è stato corretto un problema sulla USB e viene fornita nuova documentazione a corredo.

Arduino extreme: in questa versione vengono usati più componenti a montaggio superficia-

Modulo Arduino NG.

le. I connettori sono di tipo femmina a differenza delle prime versioni. Sono installati due LED sulle linee TX ed RX per monitorare il traffico della comunicazione.

Arduino NC REV.C.

Arduino extreme V2: è fatta come la versione precedente, ma con piano di massa sul PCB.

Arduino NG: è la New Generations di arduino ed utilizza il convertitore USB-Seriale di FTDI FT232RL, che richiede meno componenti esterni dell'FT232BM. Inoltre ha un LED incorporato sul pin 13. La versione plus viene fornita con un ATMega 168 invece di un ATMega8.

Arduino NG REV.C: la versione C di NG non ha il LED incorporato collegato al pin 13, ma semplicemente una resistenza da 1 kohm; il LED può quindi essere connesso all'esterno senza aggiunta di ulteriori componenti.

Arduino DIECIMILA: è dotato di interfaccia di programmazione USB e di un ATmega168 in un package DIL28. Il reset può avvenire indifferentemente via software o con pulsante sulla scheda. Viene usato uno stabilizzatore di tensione e l'alimentazione può avvenire indifferentemente via USB o dall'esterno (non serve che sia stabilizzata). Un polyfuse protegge la linea alimentazione dell'USB. Inoltre è presente un LED collegato sul pin 13, utile per i primi esperimenti.

Arduino DUEMILANOVE. È la versione aggiornata della diecimila: rispetto ad essa viene eliminato il selettore per l'alimentazione, in quanto uno switch interno commuta in automatico tra alimentazione USB o esterna. Viene eliminata la funzione di autoreset eventualmente ripristinabile con un jumper.
Dal 1° marzo 2009, il Duemilanove viene fornito con il nuovo microcontrollore ATmega328p anziché il ATmega168.

Arduino Mini: è la versione in miniatura facente uso di un ATmega168 a montaggio superficiale.

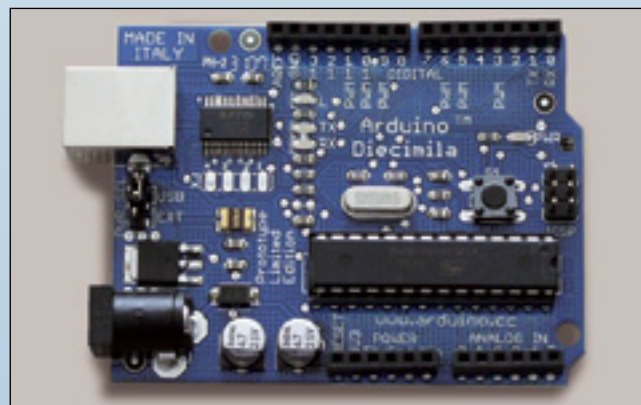
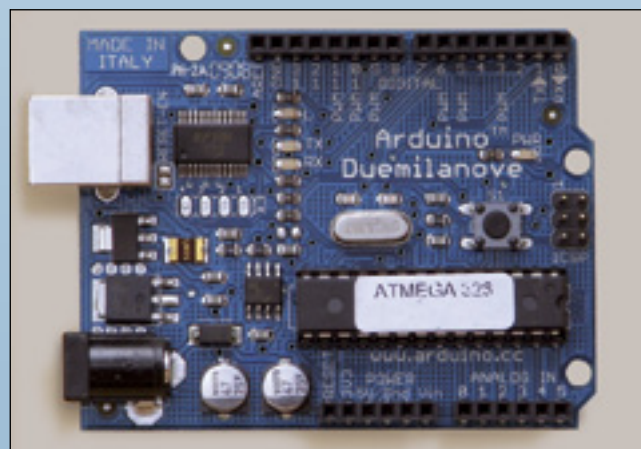
Arduino Nano: versione ancora più piccola della Mini, utilizzante lo stesso controller ATmega168 SMD e alimentata tramite USB.

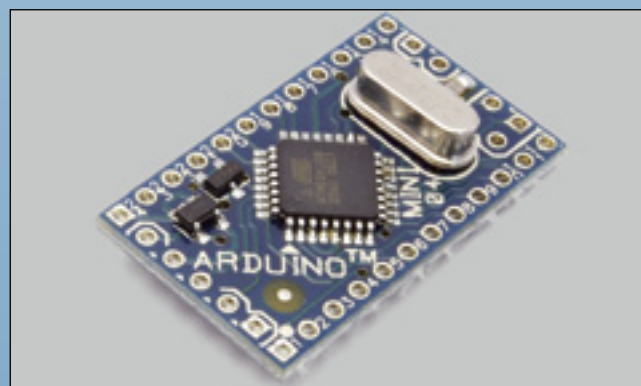
LilyPad Arduino: si tratta di un progetto minimalista per applicazioni "indossabili" basate sullo stesso ATmega168 SMD.

Arduino BT: dotato di interfaccia di programmazione Bluetooth e basato su un microcontrollore ATmega168.

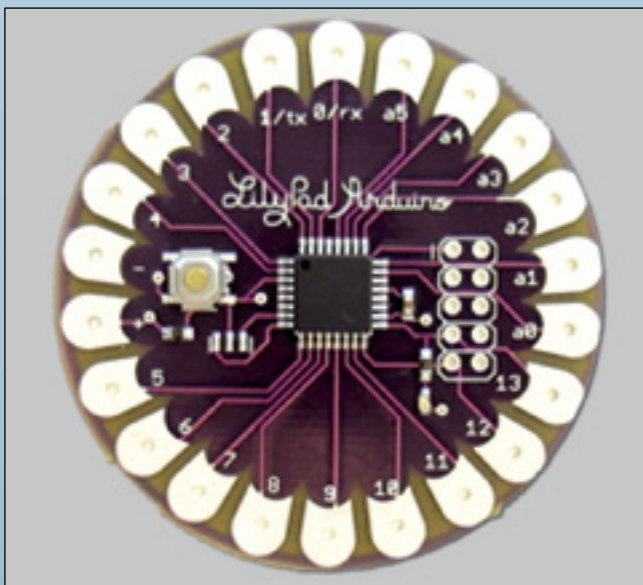
Arduino Mega: si caratterizza per il fatto che impiega un processore ATmega1280 a montaggio superficiale per la gestione di I/O e memoria addizionale.

Modulo Arduino Diecimila.

Gli schemi hardware di Arduino vengono distribuiti, in modo da poter essere utilizzati nei termini legali, con una licenza Creative Commons Attribution Share-Alike 2.5, e sono disponibili sul sito ufficiale Arduino. Per alcune versioni della scheda sono disponibili anche il layout e i file di produzione.
Il codice sorgente per l'Ambiente di sviluppo integrato e la libreria residente sono disponibili, e concessi in uso, secondo i termini legali contenuti nella licenza GPLv2.
La GNU General Public License è una licenza per software libero. È comunemente indicata con l'acronimo GNU GPL o semplicemente GPL.
Contrariamente alle licenze per software proprietario, la GNU GPL assicura all'utente libertà di utilizzo, copia, modifica e distribuzione. La GPL ha incontrato un gran successo fra gli autori di software sin dalla sua creazione, ed è oggi la più diffusa licenza per il software libero.
Come ogni licenza software, la GPL è un documento legale associato al programma rilasciato sotto tale licenza. Come tutte le licenze di software libero, essa concede ai licenziatari il permesso di modificare il programma, di copiarlo e di ridistribuirlo con o senza modifiche, gratuitamente o a pagamento. Rispetto alle altre licenze di software libero, la GPL è classificabile come "persistente" e "propagativa". È "persistente" perché impone un vincolo alla redistribuzione, nel senso che se l'utente distribuisce copie del software deve farlo secondo i termini della GPL stessa. In pratica, deve distribuire il testo della GPL assieme al software e corredarlo del codice sorgente o di istruzioni per poterlo ottenere ad un costo nominale.
Questa è la caratteristica principe della GPL, il concetto ideato da Richard Stallman e da lui battezzato copyleft. Il suo scopo è di mantenere libero un programma una volta che esso sia stato posto sotto GPL, anche se viene migliorato correggendolo e ampliandolo.
È "propagativa" perché definisce nel testo

Un esemplare di Arduino 2009.

una particolare interpretazione di "codice derivato", tale che in generale l'unione di un programma coperto da GPL con un altro programma coperto da altra licenza può essere distribuita sotto GPL.
Sia la scheda originale che i suoi cloni fanno uso di shields, ovvero di espansioni alla

Modulo Arduino Mini.

Il modulo Arduino Nano.

Arduino base, realizzate con schede a circuito stampato che possono essere collocate al di sopra della Arduino, inserendosi nei connettori normalmente forniti. Esistono espansioni dedicate a varie funzioni, dal controllo motorio,

Il coreografico LilyPad Arduino.

al breadboarding (prototipizzazione).
Tutta la documentazione originaria di riferimento costantemente aggiornata è presente sul sito ufficiale in lingua inglese **http://arduino.cc/en.** Esiste anche la versione in italiano, ma non è costantemente aggiornata; la trovate

Il modulo Arduino BT.

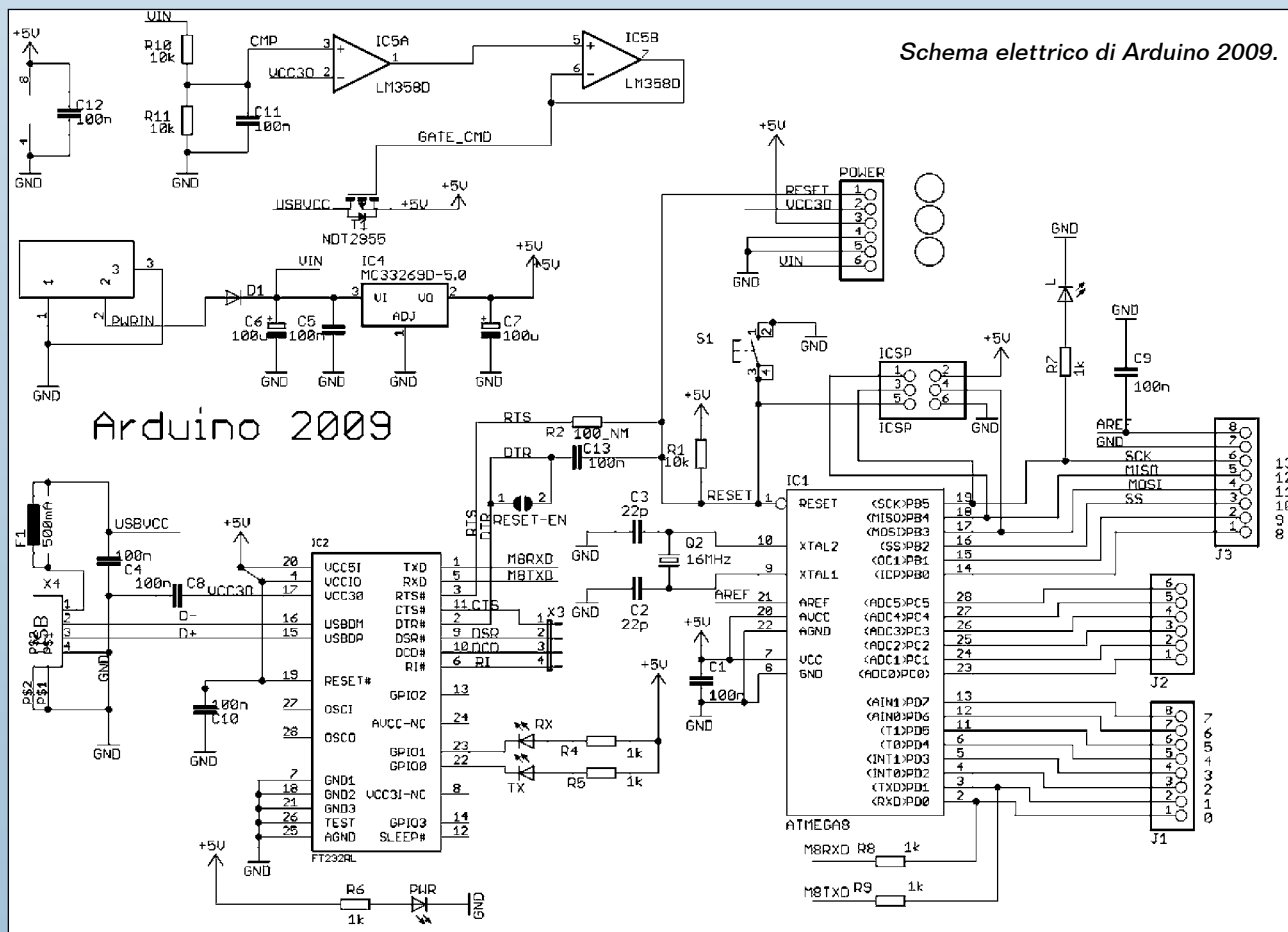
su **http://arduino.cc/it** . All'interno di questo sito potete navigare attraverso alcuni link per accedere alla sezione hardware, alla sezione software, al forum o ai tutorial; i link sono i seguenti:
- http://arduino.cc/en/Tutorial/Blink;
- http://arduino.cc/en/Main/Hardware;
- http://arduino.cc/it/Main/FAQ.

LA SCHEDA ARDUINO DUEMILANOVE

Adesso che abbiamo appreso il senso del progetto Arduino, entriamo nel dettaglio di uno dei prodotti: per la precisione, della scheda Duemilanove, che risulta essere la più recente al momento in cui scriviamo e quella che meglio rappresenta la filosofia Arduino e ben si presta ad un po' di didattica.
Analizziamone subito lo schema elettrico: la scheda Arduino Duemilanove ("2009") è basata su di un microprocessore ATmega328; in essa sono presenti 14 piedini input/output digitali (di cui 6 utilizzati come uscite PWM), 6 ingressi analogici, un oscillatore con quarzo a 16 MHz, una connessione USB, un ingresso per l'alimentazione, un ICSP header (In-Circuit Serial Programming) ed un pulsante di reset.
Tutto ciò che è necessario per supportare il microprocessore è contenuto nella scheda. Per iniziare a lavorare con essa è sufficiente connetterla ad un computer con un cavo USB oppure tramite un alimentatore AC-DC o una batteria.

Alimentazione

La scheda Arduino Duemilanove può essere alimentata tramite la porta USB del PC, oppure con un alimentatore esterno; la sorgente di alimentazione è selezionata automaticamente.
L'alimentazione esterna (non USB) può arrivare da un alimentatore AC/DC non stabilizzato (con plug da 2,1 mm) oppure da una batteria (usare i contatti Vin e GND per la connessione).
E' consigliabile alimentare la scheda con una tensione esterna da 7 a 12 volt in quanto con un potenziale inferiore il regolatore interno non riesce a fornire i 5 volt necessari mentre, con una tensione superiore, si rischia il surriscaldamento.

Schema elettrico di Arduino 2009.

I piedini di alimentazione risultano disponibili sul connettore POWER e sono elencati qui di seguito:

- VIN = tensione di alimentazione esterna; può essere fornita tramite questo pin (in alternativa al plug) oppure prelevata per alimentare una scheda esterna;
- 5V = è l'alimentazione stabilizzata per il microcontrollore e i suoi componenti. Proviene dal VIN attraverso il regolatore della scheda, oppure dalla USB o da un'altra linea di alimentazione fissa a 5 V.
- 3V3 = tensione di 3,3 volt generata dal chip FTDI della scheda; da essa non è possibile prelevare più di 50 mA. La linea corrispondente può essere usata per alimentare schede esterne.
- GND = piedino di massa (zero volt).

Memoria

La ATmega328 dispone di 32 kB di memoria flash per caricare il codice (di cui 2 sono utilizzati per il bootloader). Ha inoltre 2 kB di SRAM e 1 kB di EEPROM (che può essere letta e scritta con la libreria EEPROM).

Ingressi e uscite

Ciascuno dei 14 piedini digitali della Duemilanove può essere utilizzato sia come ingresso che come uscita, utilizzando le funzioni pinMode(), digitalWrite(), e digitalRead() ed operando a 5 volt. Ogni piedino gestisce al massimo 40 mA ed ha una resistenza di pull-up (disconnessa di default) da 20÷50 kohm. Inoltre, alcuni piedini hanno delle funzioni specializzate che descriviamo di seguito.

- Serial: 0 (RX) e 1 (TX). Utilizzati per ricevere (RX) e trasmettere (TX) in TTL dati seriali. Questi piedini sono connessi ai corrispondenti piedini del chip seriale FTDI USB-to-TTL.
- Interruttori esterni: 2 e 3. Questi piedini possono essere configurati come ingressi digitali.
- PWM: 3, 5, 6, 9, 10, e 11. Fornisce un'uscita a 8-bit PWM con la funzione analogWrite().
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Questi piedini supportano la comunicazione SPI, che, sebbene implementata a livello hardware, non è al momento inclusa nel linguaggio Arduino.
- LED: 13. Connesso al piedino digitale 13

Microprocessore	ATmega328
Tensione operativa	5 V
Tensione di alimentazione (raccomandata)	7÷12 V
Tensione di alimentazione (limite)	6÷20 V
Piedini digitali I/O	14 (di cui 6 utilizzati come output PWM)
Piedini di ingresso analogici	6
corrente DC per i piedini di I/O	40 mA
corrente DC per il piedino a 3.3V	50 mA
Memoria Flash	16 kB (di cui 2 kB utilizzati dal bootloader)
SRAM	1 kB
EEPROM	512 byte
Clock	16 MHz

Caratteristiche dell'Arduino Duemilanove.

è montato sulla scheda un LED. Quando questo ha valore HIGH, il LED è acceso, quando il piedino è LOW, il LED è spento.

La Duemilanove ha sei ingressi analogici, ognuno dei quali con una risoluzione di 10 bit (ovvero 1.024 valori differenti). Per impostazione predefinita essi accettano in ingresso una tensione tra GND e 5 V, sebbene sia possibile cambiare il limite superiore utilizzando il piedino AREF. Inoltre, alcuni piedini hanno delle funzioni specializzate; si tratta di:

- 4 (SDA) e 5 (SCL); riguardano un bus I²C e supportano la comunicazione I²C (TWI) utilizzando la libreria Wire (la documentazione del caso si trova sul sito della Wiring).

Ci sono poi altri due piedini specializzati, che sono:

- AREF; tensione di riferimento per gli ingressi analogici; viene utilizzato con analogReference().
- Reset; portata a livello basso permette di resettare il microprocessore (inizializzazione).

A questo proposito conviene consultare il data-sheet dell'ATmega168.

Comunicazione

La Arduino Duemilanove ha una serie di funzioni utili alla comunicazione con un computer, un'altra scheda Arduino, o altri microprocessori. L'ATmega328 dispone di un modulo di comunicazione seriale UART TTL-compatibile (5 V) accessibile dai piedini digitali 0 (RX) e 1 (TX). Un integrato FTDI FT232RL sulla scheda canalizza questa comunicazione seriale sulla USB; i driver forniti dalla FTDI (disponibili con il software Arduino) provvedono a creare una porta COM virtuale utilizzabile dal software presente sul computer. Il software Arduino include un monitor seriale che permette di spedire o ricevere dalla scheda Arduino semplici dati di testo.
Una libreria SoftwareSerial permette la comunicazione seriale su qualunque dei piedini digitali della Duemilanove. Il microcontrollore ATmega328 supporta anche la comunicazione I²C (TWI) e la SPI. Il software Arduino include una libreria Wire per semplificare l'uso del bus I²C; la documentazione del caso si trova sul sito Wiring, che potete consultare per dettagli. Per sapere come utilizzare la comunicazione SPI communication, consultate i dati tecnici della ATmega328.

Programmazione

Il microcontrollore ATmega328 della scheda Arduino Duemilanove ha già un bootloader pre-caricato, che permette di caricare nuovo codice senza la necessità di uno specifico programmatore esterno; comunica utilizzando il protocollo originale STK500. Si può, naturalmente, evitare l'utilizzo del bootloader e programmare la ATmega328 attraverso il connettore ICSP (*In Circuit Serial Programming*).

Inizializzazione automatica (Software)

Il microcontrollore può essere resettato sia a livello hardware con l'apposito pulsante sulla scheda, sia via software al caricamento di un programma. Una delle linee di controllo del flusso hardware (DTR) della FT232L è connessa alla linea di reset della ATmega328 attraverso un condensatore da 100 nanofarad. Quando questa linea viene alimentata con una tensione troppo bassa, la linea di reset assume un livello sufficiente ad impartire il reset della scheda. Il software Arduino utilizza questa caratteristica per caricare codice semplicemente premendo il pulsante di caricamento (upload). Ciò significa che il bootloader può avere un timeout più breve, mentre l'abbassamento del DTR può essere coordinato con l'inizio del caricamento.
Questa configurazione ha altre implicazioni: quando la Duemilanove è connessa ad un Mac o ad un PC con sistema operativo Linux, ogni volta che viene fatta una connessione via software (attraverso la porta USB) essa si re-inizializza. Per il mezzo secondo (circa)

successivo, sulla Duemilanove sarà attivato il bootloader.
Poiché la programmazione è stata fatta con la caratteristica di ignorare i dati non formattati correttamente (ad esempio, tutto ciò che non ha a che fare con il caricamento di un nuovo codice) intercetterà i primi pochi byte dei dati spediti alla scheda quando viene aperta una connessione. Se all'Arduino è connessa una scheda aggiuntiva che utilizza questa porta dati per comunicare con il microcontrollore, è importante che il software con cui comunica attenda circa un secondo dopo l'apertura della connessione e prima della trasmissione di qualunque dato. La Duemilanove contiene una pista che permette di disabilitare l'autoreset: è sufficiente ripristinare il contatto (che sullo stampato viene identificato come "RESET-EN") per ripristinare la funzione.

Protezione da sovratensione della USB
La Arduino Duemilanove ha un fusibile autoripristinante che protegge la porta USB del computer da cortocircuiti ed eccessiva tensione di alimentazione. Anche se la maggior parte dei computer già prevede una protezione interna, il fusibile fornisce un ulteriore livello di tutela. Se più di 200 mA attraversano la porta USB, il fusibile interrompe automaticamente la connessione fino a quando il cortocircuito o il picco non venga meno.

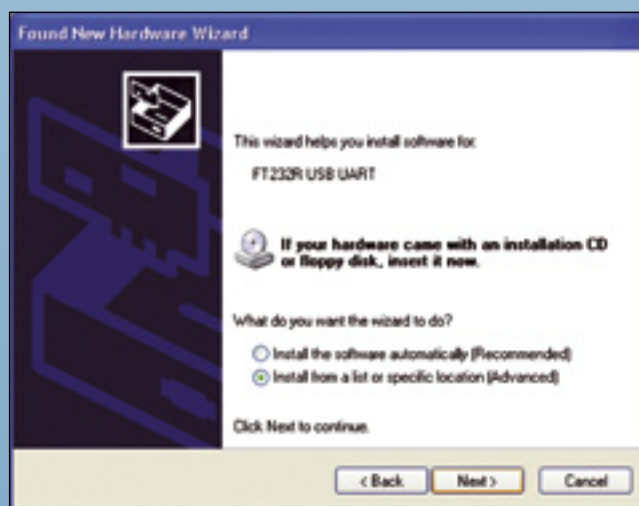
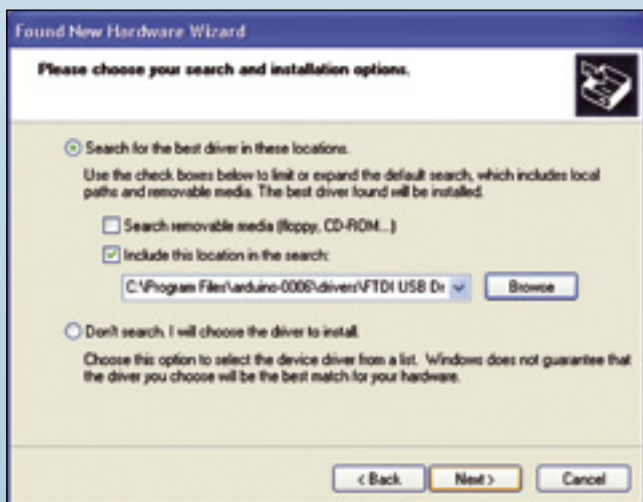
Caratteristiche fisiche
Il PCB della scheda Duemilanove misura 6,8 per 5,33 cm e riporta tre fori per il fissaggio della scheda ad una superficie o ad un contenitore. Il connettore USB e il plug per l'alimentazione escono leggermente dal profilo dello stampato.

Approfondimenti sull'hardware
Una prima fonte di alimentazione può essere applicata al plug al quale fa capo un diodo a protezione dall'inversione di polarità ed uno stabilizzatore di tensione a 5 volt. L'alimentazione giunge anche tramite il connettore USB, ma solo se non è presente l'alimentazione primaria; infatti il circuito composto dall'operazionale IC5 provvede a disabilitare l'alimentazione dalla USB, portando in OFF il MOSFET NDT2955 quando sente la presenza dell'alimentazione primaria letta tramite le

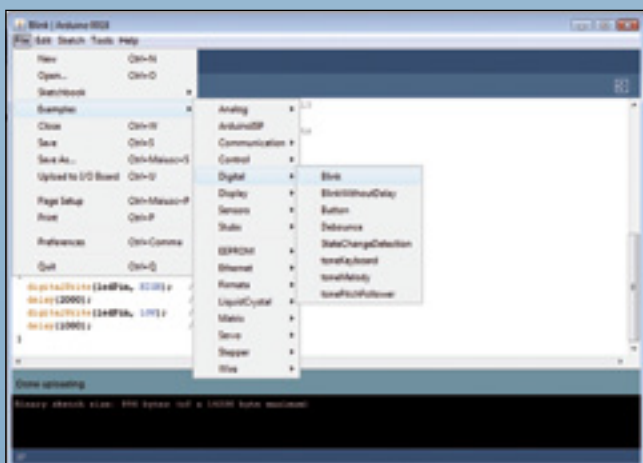
Contenuto del file compresso scaricabile da Internet.

resistenze R10 ed R11.
Il clock è ottenuto tramite un quarzo a 16 MHz,

Avvio schermata richiesta driver.

che stabilisce l'intervallo di tempo per l'esecuzione di una istruzione, in quanto quasi tutte

Impostazione manuale del percorso dei driver.

Percorso dei driver USB.

Completamento dell'installazione dei driver.

le istruzioni necessitano di un ciclo di clock per la loro esecuzione. Nei microcontrollori Microchip, ad esempio, sono necessari 4 impulsi di clock per eseguire un'istruzione e quindi il numero di istruzioni eseguibili in

Il programma di esempio "LED blink".

un" secondo (MIPS) equivale ad un quarto della frequenza di clock.
Notiamo dallo schema che tutti i segnali sono disponibili nei vari connettori, quindi, oltre agli IN/OUT, disponiamo del segnale seriale in uscita dall'FT232, dei segnali per il programmatore seriale esterno e delle tensioni di alimentazione. A completare la scheda, il LED PWR, acceso in presenza di alimentazione.

Come si usa

Ma entriamo nel cuore dell'applicazione, ovvero del software di sviluppo necessario alla programmazione del microcontrollore, senza cui la scheda Arduino, appena acquistata, non fa assolutamente nulla. Per poterla programmare da remoto senza alcun specifico programmatore, il microcontrollore viene fornito preprogrammato con uno specifico bootloader che instraderà in modo corretto il firmware nell'apposita area di memoria EEPROM durante la fase di programmazione. Faremo riferimento alla scheda Arduino duemilanove, ma nello stesso modo potremo operare con altre schede Arduino (tipo USB) e cloni, purché provvisti di interfaccia USB.
Siccome la scheda dispone di un LED, come primo esperimento lo faremo lampeggiare.
La Arduino Diecimila (e la Arduino NG originaria) hanno una resistenza e un LED integrati collegati sul pin 13. Nelle schede Arduino NG Rev. C e precedenti il pin 13 ha comunque una resistenza integrata, ma bisogna provvedere a collegare un LED esterno. In questo caso è necessario connettere il polo positivo (il reoforo più lungo) del LED al pin 13 e il polo negativo (quello corto) alla massa (marcata con "GND"); per non sbagliare, ricordate che i LED sono normalmente piatti dalla parte del catodo (negativo).
Vediamo ora, per passi, come improntare la nostra primissima applicazione.

1. Procurare il materiale

Per prima cosa dobbiamo procurarci una scheda Aduino Duemilanove ed un cavo USB con connettore standard A-B, che è quello normalmente utilizzato per collegare una stampante USB al computer.

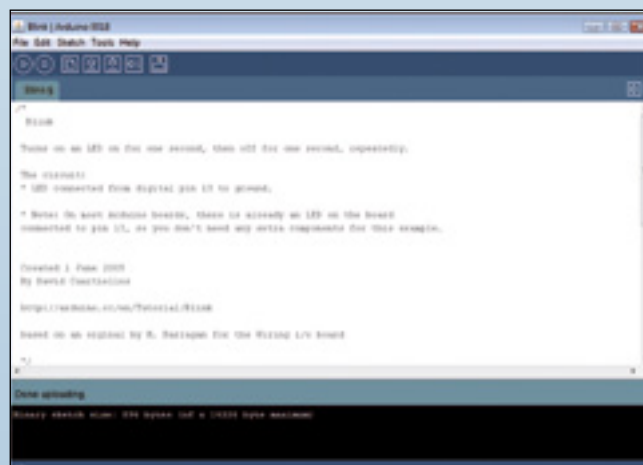
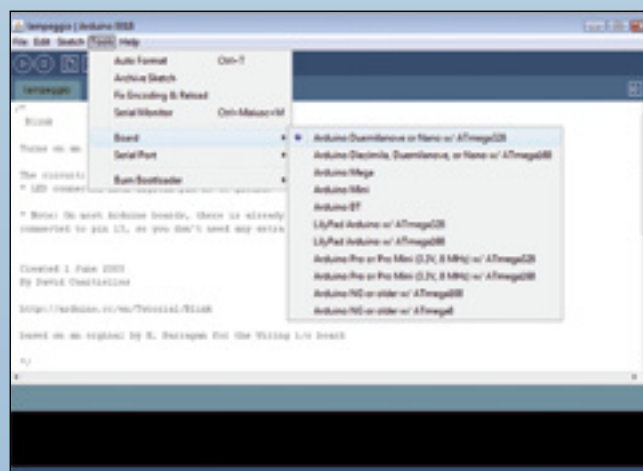
2. Scaricare il software Arduino

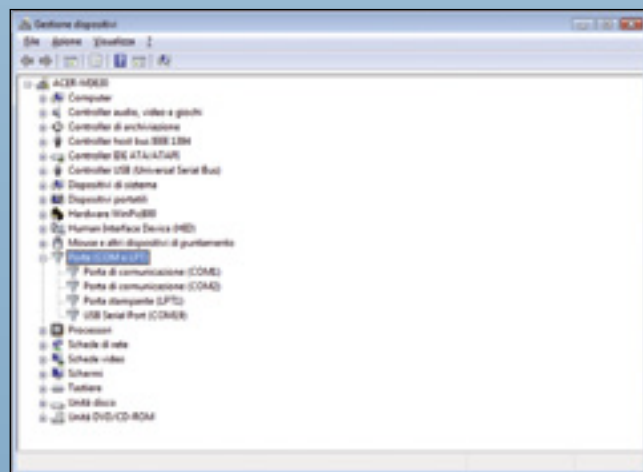
Per programmare la scheda Arduino è necessario disporre dell'ambiente software

Arduino. Facendo riferimento al sito ufficiale Arduino, scarichiamo il software relativo all'ambiente di sviluppo; al momento in cui scriviamo è disponibile la versione per windows *arduino-0018.zip*, quella per MAC *arduino-0018.dmg* e quella per linux *arduino-0018.tgz*. Nella successiva descrizione faremo riferimento all'installazione in ambiente Windows. Decomprimete il file scaricato assicurandovi di conservare la struttura delle cartelle.
Nelle varie cartelle sono compresi, oltre al sistema di sviluppo, tutti i file java necessari, i driver per FT232 e gli esempi del caso.

3. Connettere la scheda
Come prima applicazione suggeriamo di alimentare la scheda direttamente dalla USB; per fare questo è sufficiente inserire il cavo tra la porta USB del PC e la scheda. Non ci sono jumper o deviatori da impostare, quindi il LED di alimentazione (PWR) deve illuminarsi. Appena inserita la scheda, il sistema operativo Windows inizia l'installazione dei driver; con Vista questo passaggio è automatico, in quanto esso ricerca autonomamente i drive e li installa, operazione che richiede alcuni secondi. Per sistemi operativi più vecchi la procedura avviene manualmente.
Una volta aperta la finestra di dialogo della richiesta dei driver bisogna fare clic sul pulsante sfoglia e specificare il percorso in cui trovare i driver: nel nostro caso è la cartella FDT USB driver contenuta nei file di Arduino. Fatto ciò si deve avviare l'installazione dei driver.

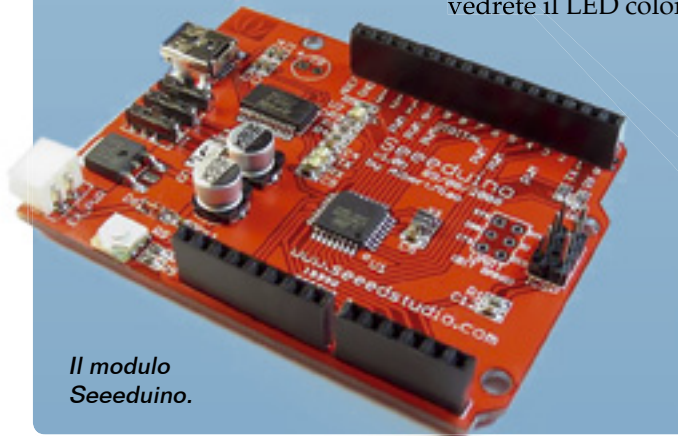
4. Avviare il software
Avviamo il software, aprendo la cartella Arduino appena decompressa e facendo doppio clic sull'icona dell'eseguibile Arduino.

5. Caricare un programma
Apriamo il programma di esempio LED blink: **ARDUINO-0018/Examples/Digital/Blink.pde**.
Di lato ecco come appare il codice del programma di esempio che farà lampeggiare il LED. Selezionate la scheda Arduino dall'elenco disponibile in TOOL-BOARD.
Selezionate la COM alla quale è connessa la scheda. Questa indicazione viene anche riportata durante l'installazione dei driver, ma

Il codice del programma LED Blink.

Menu di scelta della scheda Arduino.

può essere ricavata leggendo dalle periferiche installate tramite il percorso: Computer- (pulsante destro proprietà) - gestione dispositivi- porte COM & LPT - USB serial PORT.
Assicuratevi che sia selezionata la voce "Arduino duemilanove" nel menu Tools > Board.
Ora basta semplicemente fare clic sul pulsante

Ricerca della porta COM virtuale assegnata ad Arduino.

Definizione della COM virtuale nel programma.

Il pulsante di compilazione.

"Upload" nell'interfaccia grafica del programma. Dopo qualche secondo dovrebbero lampeggiare molto velocemente i LED RX e TX sulla scheda.
Significa che la scheda sta comunicando con il computer. Se la procedura è andata a buon fine apparirà il messaggio "Done uploading" nella barra di stato.
Notate che, se state utilizzando una scheda Arduino Mini, NG, o altri tipi di schede, dovrete premere fisicamente il loro pulsante di reset appena prima di fare clic su "Upload" nel software; nelle altre schede il reset è automatico.
Il software di sviluppo non ha la necessità di creare file intermedi, perché partendo direttamente dal codice sorgente crea, immediatamente il codice macchina da inserire nel microcontrollore; il tutto, in un unico passaggio. Esiste comunque il pulsante di compilazione che provvede a verificare la correttezza del codice. Qualche secondo dopo il termine del processo di upload, vedrete il LED color ambra lampeggiare sulla scheda.
Congratulazioni! Se lo vedete, avete una scheda Arduino connessa e funzionante.

Il modulo Seeeduino.

CLONI DI ARDUINO

Per il fatto che è possibile per terze parti creare una propria Arduino compatibile con il software originale, sono disponibili in commercio diversi cloni. Benché i progetti hardware e software siano resi disponibili con licenze copyleft, gli sviluppatori hanno espresso il desiderio che il nome "Arduino" (o suoi derivati) venga riferito solo al prodotto originale e non sia usato per indicare opere derivate senza il permesso. Il documento che esprime la policy ufficiale sull'uso del nome "Arduino" mette l'accento su come il progetto sia aperto ad incorporare lavori altrui nel prodotto ufficiale. Quale conseguenza di queste convenzioni sulla protezione del nome, molti prodotti simili ad Arduino sono presenti sul mercato ma con nome diverso dall'originale, come Freeduino o Seeeduino. Il nome non è però inteso come un marchio commerciale ma è liberamente utilizzabile da chiunque lo desideri. Tra le schede disponibili sul mercato, la SeeeduinoV2.12 (completamente assemblata con componenti SMD) propone una valida alternativa all'originale con alcune differenze, tra cui la possibilità di programmazione via USB con connettore micro ed alimentazione esterna tramite connettore JST.

CARATTERISTICHE DI SEEEDUINO

Seeeduino v2.12 è una scheda compatibile con Arduino Diecimila e basata sul microcontrollore ATmega168. La pin-out, i fori di fissaggio e le dimensioni sono compatibili al 100 % con quelle di Arduino Diecimila. La scheda dispone di 14 I/O (di cui sei possono essere utilizzati come uscite PWM), 8 ingressi analogici, 16 kB di memoria flash, 1 kB di SRAM e 512 byte di memoria EEPROM. Rispetto alla scheda Arduino Diecimila, presenta alcune differenze di seguito elencate.

1. Gli stessi ingressi e uscite sono disponibili su due connettori differenti.
2. Il microcontrollore ATmega168 versione DIP è stato sostituito con la versione SMD; ciò ha permesso di ottenere più spazio sul PCB, semplificando l'inserimento delle schede dei vostri prototipi sui connettori

del Seeduino.
3. Per consentire un facile accesso, tutti i pulsanti e gli interruttori sono vicini ai bordi del PCB.
4. A causa delle sue dimensioni ingombranti, il connettore USB tipo B è stato sostituito con un connettore Mini USB.
5. A causa delle sue dimensioni ingombranti, la presa di alimentazione (jack da 3,5 mm) è stata sostituita con connettore JST a 2 poli.
6. Dispone di LED indicatore presenza alimentazione e di Reset vicino al pulsante RST.
7. Ha la funzione di Auto-reset selezionabile.
8. Dispone di interruttore di selezione per tensione a 3,3 V o 5 V.
9. Interfaccia UART per FTDI232 che permette di trasferire il bootloader senza la necessità di utilizzare un cavo ISP.
10. Sono state aggiunti 2 ingressi ADC.
11. Facile connessione I²C e sensori analogici.
12. Possibilità di alimentazione diretta del Seeeduino a 5 Vcc (Attenzione: Usare solo 5 V) mediante ingresso supplementare.
13. Il microcontrollore ATmega168 versione DIP è stato sostituito con la versione SMD; questo ha permesso di ottenere più spazio sul PCB.
14. Dispone di connessione USART.
15. Riga supplementare di pin a saldare. È così possibile utilizzare un connettore femmina o maschio a propria scelta.
16. Dispone di un regolatore di tensione da 3,3 V in grado di fornire più corrente (150 mA) rispetto ai 50 mA forniti dall' FT232.

Seeeduino V3.28

Seeeduino è una scheda compatibile con Arduino duemilanove basata sul microcontrollore ATmega328. Differisce dal modello Seeeduino V2.12 perchè dispone di maggior memoria flash, EEPROM e SRAM. Per quanto riguarda l'utilizzo, essa non di discosta molto dalla versione originale.
Sulla scheda sono disponibili tre piccoli deviatori con le seguenti funzioni:
- seleziona se l'alimentazione giunge dalla USB o da fonte esterna; per la prima applicazione impostiamo l'alimentazione da USB, così da non doverci procurare ulteriori alimentatori;

Funzionalità della scheda Seeeduino.

- seleziona l'alimentazione della logica a 5 V o 3,3 volt; 5 volt vanno benissimo come inizio e i 3,3 V potrebbero essere utili se si intende usare la scheda per alimentare un circuito esterno funzionante a tale tensione;
- il terzo deviatore seleziona la modalità di reset; impostandola su automatico la scheda si resetterà in automatico non appena sarà caricato il firmware.
Per la prima applicazione impostiamo tutti i deviatori con la levetta rivolta verso l'interno della scheda. La scheda viene connessa al PC con il solito cavo USB (con connettore micro dal lato scheda); la procedura di avvio e di programmazione è identica alla scheda Arduino originaria descritta in precedenza.

Seeeduino V3.28.

Funzioni degli interruttori a slitta di Seeeduino.

Conoscere e usare Arduino

dell'ing.
MIRCO
SEGATELLO

Impariamo ad utilizzare le shield e mettiamo in pratica alcuni semplici programmi, chiamati "sketches", che permettono di sfruttare le funzioni della Danger Shield.

Nella prima puntata abbiamo presentato il sistema di sviluppo italiano per microcontrollori Atmel denominato Arduino descrivendo i vari hardware messi in commercio nelle loro numerose versioni, ma anche descrivendo l'ambiente di sviluppo e riportando i link di riferimento per i tutorial ed il download del software. Il primo esempio, semplicissimo, descriveva l'accensione di un LED e aveva lo scopo di farci prendere confidenza con il sistema di sviluppo in modo semplice e graduale. In questa seconda puntata prendiamo in considerazione la realizzazione di applicazioni più complesse, ma sempre idonee alla comprensione di questo interessante sistema di sviluppo: per la precisione, ci occuperemo di alcune *shield* . Ricordiamo che esse non sono altro che appositi circuiti, i quali, inserendosi direttamente nei connettori della scheda principale (quella con a bordo il microcontrollore) ne espandono le funzionalità aggiungendo hardware più specifico. Facciamone ora una breve carrellata giusto per avere conoscenza dei prodotti oggi disponibili in commercio e valutare le possibili applicazioni. Il sistema è comunque costantemente in sviluppo e nuove ed interessanti funzioni vengono implementate e rese disponibili a tutti; infatti non dimentichiamo che, secondo la licenza concessa dallo staff che ha sviluppato Arduino, chiunque può sviluppare espansioni sia per un uso personale, sia per fini commer-

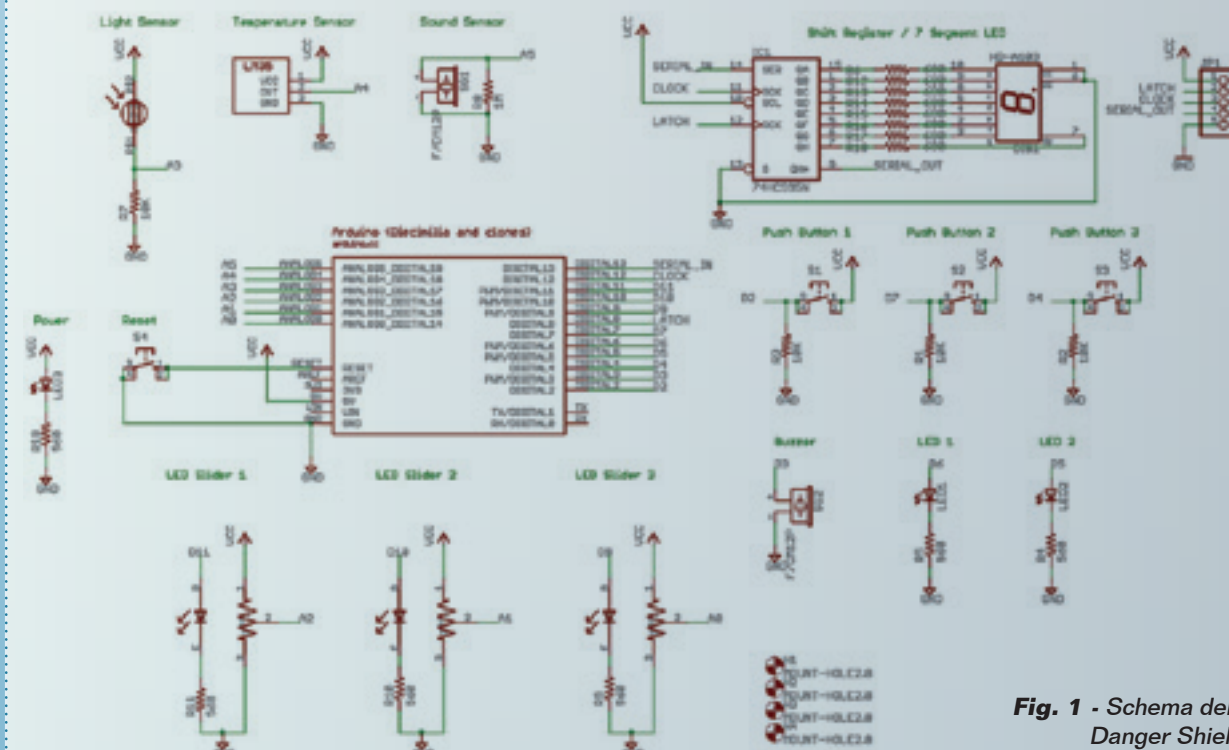



***Fig. 1** - Schema della Danger Shield.*

ciali. Le shield che analizziamo sono elencate e sommariamente descritte qui di seguito.

- **Protoshield**: è una piastra sperimentale (58,50 x 82,70 mm) per piccole applicazioni, realizzata appositamente per le schede Arduino o Seeeduino; permette di avere un numero maggiore di piazzole su cui montare i componenti. Alcuni piazzole sono predisposte per montare un connettore USB tipo B, un mini potenziometro da stampato, pulsanti, LED, ecc. Dispone di piazzole riservate al montaggio di connettori per UART, ISP e I²C.
- **Danger Shield**: montata sopra una scheda Arduino o Seeeduino, permette di testare i vari ingressi/uscite, grazie ad una serie di componenti elettronici. Il KIT contiene 3 Slider lineari con LED integrati, 4 pulsanti, 3 LED indicatori, 1 Buzzer, 1 Sensore di temperatura.
- **Motor Shield**: basata sul chip L298, permette di controllare direzione e velocità di 2 motori DC con una corrente massima di 2 ampere ciascuno. Alimentata direttamente dalla linea Vin di Arduino Duemilanove o Seeeduino, ogni uscita dispone di un LED blu e uno giallo per indicare la direzione di rotazione del motore. Tutte le linee di uscita del chip L298 sono protette da un diodo.
Ethernet Shield: è un dispositivo basato sul chip ethernet Wiznet W5100, che permette di connettere una scheda Arduino ad una LAN utilizzando la libreria Ethernet library. Ardui-no Ethernet Shield supporta fino a quattro connessioni socket simultanee.
- **Wiznet Shield**: si tratta di un modulo basato sul chip W5100 e dotato di connettore ethernet completo di filtri magnetici e 2 LED per la segnalazione di stato. Supporta i protocolli hardware Ethernet: DLC, MAC e internet: TCP, IP Ver.4, UDP, ICMP, ARP, PPPoE, IGMP. Funziona in modalità sia full-duplex che half-duplex. Questo modulo è la scelta ideale per gli utenti che desiderano dotare i loro dispositivi di connessione Internet.
- **GPS Shield**: permette di dotare la scheda Arduino di un modulo ricevitore GPS. Grazie alle librerie disponibili in rete, sono facilmente realizzabili applicazioni come data-logger, localizzatori ecc. Il GPS Shield è dotato di un connettore per il montaggio

del ricevitore GPS 8160-EM406A e di una piccola area riservata alla prototipazione.

- **Tellymate Shield**: collegato ad una scheda Arduino e, utilizzando il comando "Serial. println()", permette di inviare semplici testi e grafici sul vostro televisore. La velocità di trasmissione può essere selezionata tramite ponticelli.
- **UARTSBV22**: si tratta di un convertitore USB-seriale che permette di dotare i vostri progetti di connessione USB. Viene utilizzato per gestire la scheda RAIMBOWDUINO e gli strip di cui è dotato permettono una facile integrazione con altre schede. La scheda offre la possibilità di montare un modulo XBee o Bluetooth, permettendo in questo modo di ottenere una connessione wireless con il PC.
- **SD Card shield**: è una piccola scheda dotata di slot per SD-Card e permette ai dispositivi come Arduino e Seeeduino di leggere e scrivere le SD-Card con apposite librerie disponibili gratuitamente. L'unità è dotata di un piccolo deviatore per selezionare la fonte di alimentazione dell'SD card: tramite pin o 3,3 V della scheda Arduino/ Seeeduino.

In questo articolo ci occuperemo essenzialmente della Danger Shield e dalla sua programmazione. Questa scheda, reperibile in kit di montaggio (ad esempio su *www.futurashop.it*) è l'ideale per le prime applicazioni, e permette di prendere confidenza con l'hardware ed il software del sistema Arduino. All'interno della confezione non sono fornite le istruzioni (per altro disponibili in Internet) quindi per la sua installazione dovete seguire le indicazioni seguenti. Dal suo schema elettrico (**Fig. 1**) potete notare la presenza di un LED (LED3) di segnalazione di presenza dell'alimentazione, i LED1 e LED2 usabili a piacimento per segnalazioni varie, tre potenziometri a slider che forniranno una tensione analogica in ingresso al microcontrollore, un buzzer ed un sensore di tocco ottenuto tramite un secondo buzzer, tre pulsanti funzionanti in logica positiva (premuti, forniscono uno logico) e, infine, un display a LED a sette segmenti controllato da uno shift-register tipo 74HC795. La corrispondenza tra le periferiche e i pin del microcontrollore è illustrata nella **Tabella 1**. La corrispondenza tra la sigla delle resistenze ed il loro valore è riportata nella **Tabella 2**. Sullo stampato è riportata la serigrafia per tutti i componenti, quindi non dovrebbero esserci difficoltà nel trovare l'ubicazione di ciascun elemento; per aiutarvi fate riferimento alle foto dell'articolo. Come al solito, iniziate dalle resistenze montate in piano ed a seguire collocate gli altri componenti, in ordine di altezza. Le resistenze del display a LED sono invece posizionate in verticale; fate molta attenzione perché lo spazio a disposizione è molto esiguo. Per ultimi saldate i potenziometri. Fate estrema attenzione alla giusta inserzione dei LED e del display, del buzzer e del sensore di tocco. Notate una curiosità: il sensore di tocco ed il buzzer in pratica sono il medesimo componente; infatti il cicalino piezoelettrico è reversibile, in quanto se gli applichiamo corrente elettrica esso fornisce un suono, ma se non lo alimentiamo e lo sollecitiamo meccanicamente (ad esempio gli diamo un colpo sufficientemente intenso con un dito) genera una tensione elettrica. Detto ciò, vi rimane da saldare i connettori

Tabella 1

Nome periferica	Nome sullo stampato	Segnale microcontrollore
LED3	Power	Presenza alimentazione
Pulsante_1	Button 1	D2
Pulsante_2	Button 2	D7
Pulsante_3	Button 3	D4
LED2	LED2	D5
LED1	LED1	D6
Potenziometro 3	Slider3	A0
Potenziometro 2	Slider2	A1
Potenziometro 1	Slider1	A2
Fotoresistenza	LDR	A3
Sensore temperatura	Temp sensor	A4
Sensore di suono	SG1 Knock sensor	A5
LED su slider1	LED su slider1	D9
LED su slider2	LED su slider2	D10
LED su slider3	LED su slider3	D11
Buzzer	SG2 Buzzer	D3
In seriale 795	Serial	D13
Clock per 795	Clock	D12
Enable per 795	Latch	D8

Tabella 2

Resistenza	Valore
R1, R2, R3, R7	10 K
R6, R12, R13, R14, R15, R16, R17, R18	680
R4, R5, R9, R10, R11, R19	560
R8	1 M
IC1	74HC795

Tabella 3

	Verify/Compile Compila e verifica errori nel listato.
	Stop Ferma l'applicativo Serial Monitor o altra funzione attiva.
	New Crea un nuovo sketch.
	Open Per aprire uno sketch esistente negli esempi o in una cartella proprietaria.
	Save Salva l'attuale sketch aperto.
	Upload to I/O Board Compila e trasferisce il codice all'interno della Arduino board.
	Serial Monitor Apre l'applicativo serial monitor. Utile per inviare o ricevere velocemente caratteri dalla porta di comunicazione seriale.

che permettono il fissaggio alla Arduino board; nel nostro esempio abbiamo utilizzato gli strip maschi forniti nella confezione, il che ci permette di tenere la Danger board ad alcuni millimetri di distanza dal connettore USB e di alimentazione della Arduino board.
Per sicurezza ponete del nastro adesivo isolante sopra il connettore USB, così da essere certi che incidentalmente questo non vada a toccare le piste dello stampato della Danger Shield.
L'ambiente di sviluppo Arduino contiene un editor di testo, un'area per i messaggi, una toolbar con pulsanti per le funzioni più frequenti ed una serie di menu. I programmi scritti con Arduino si chiamano *Sketches* e vengono scritti nell'editor di testo, il quale, è in grado di gestire contemporaneamente più file aperti ad esempio C files (**.c extension**), C++ files (**.cpp**), oppure header files (**.h**). Tutte le classiche funzioni di copia/incolla, ricerca del testo ecc. sono state implementate, così come la messaggistica relativa agli errori ed allo stato di lavoro. La toolbar per le funzioni più frequenti comprende i comandi visibili nella **Tabella 3**. I menu riepilogati nella barra dei menu soprastante (File, Edit, Sketch, Tools, Help) permettono di accedere alle funzioni più evolute e meno frequenti, come ad esempio la selezione della scheda Arduino da noi utilizzata. Di seguito descriviamo quelle più importanti

Listato 1

```
/*
 Danger_01

 Lampeggio LED1.

 Per DangerShield su Arduino Duemilanove

 */

int ledPin =  6;    // LED connesso al pin digitale D6


void setup()   {
  // inizializza il pin del LED come uscita digitale:
  pinMode(ledPin, OUTPUT);
}

// Viene dichiarato un loop senza uscita
// Le istruzioni interne al loop vengono continuamente eseguite

void loop()
{
  digitalWrite(ledPin, HIGH);   // Accende il LED
  delay(1000);                           // aspetta un secondo
  digitalWrite(ledPin, LOW);    // spegne il LED
  delay(1000);                  // aspetta un secondo
}
```

Edit
- *Copy for Discourse*: copia il codice selezionato.
- *Copy as HTML*: copia il codice selezionato e lo rende disponibile per il trasferimento in una pagina Web.

Sketch
- *Verify/Compile*: compila e verifica gli errori nel listato.
- *Import Library*: aggiunge una libreria allo sketch attuale inserendo #include statements nel codice.
- *Show Sketch Folder*: apre la cartella contenente I file dello sketch attuale.
- *Add File...* : aggiunge un file sorgente allo sketch attuale.

Tools
- *Auto Format*: serve per formattare il codice ad esempio per renderlo "identato", cioè per applicare rientri diversi a seconda delle righe.
- *Board*: seleziona la scheda Arduino utilizzata.
- *Serial Port*: per impostare la porta seriale sulla quale la scheda è connessa.
- *Burn Bootloader*: questo menu permette di avviare la procedura di installazione del bootloader all'interno della scheda Arduino.

Questa operazione non è richiesta

con le schede arduino in quanto vengono fornite con il bootloader già pre-caricato.

Uploading
Per caricare il vostro sketch è necessario prima selezionare la corretta Arduino Board dal menu *Tools* > *Board* ed impostare la giusta porta di comunicazione, dal percorso *Tools* > *Serial Port*.

Libraries
Le librerie permettono di estendere le funzionalità di base implementate. Questo comando permette di aggiungere una libreria allo sketch tramite Sketch > Import Library menu.

Serial Monitor
Questa applicazione permette di inviare dei semplici caratteri attraverso la porta seriale verso la scheda Arduino. In alcune schede in cui è implementata la comunicazione USB, la porta seriale è solo virtuale.

Boards
Da questo menu è possibile selezionare l'unità Arduino utilizzata.

Help
Questo menu permette di accedere ad una serie di informazioni riguardanti l'ambiente di sviluppo tramite l'accesso diretto a delle pagine HTML contenute nella cartella *reference*. Proprio questa voce del menu permette di aprire una pagina HTML contenente tutte le istruzioni disponibili e cliccando sopra ognuna di esse è possibile entrare nei dettagli.

LA PROGRAMMAZIONE

Seguendo le istruzioni riportate nella prima puntata del corso sarete in grado di installare e rendere operativo il software di sviluppo. Il primo programma che realizzeremo non si discosterà molto dall'esempio già proposto il mese scorso, dato che faremo semplicemente lampeggiare il LED1 della Danger Shield, connesso al pin 6 del microcontrollore. Dopo aver avviato il software (Arduino ver. 18, nel nostro caso) potete scegliere di creare da zero un nuovo progetto oppure, in modo più veloce, utilizzare un programma simile già disponibile modificando le parti necessarie. Consigliamo di creare una nuova cartella, nel nostro caso Danger_01, nella quale potremmo inserire un programma già esistente (simile per funzione) come ad esempio il file *Blink.pde* presente nella cartella *Example/Digital/Blink*, e rinominarlo come *Danger_01.pde*.
Ricordiamo che Arduino è sì un prodotto italiano, ma per favorirne la massima diffusione sia il software che la documentazione sono scritti in lingua inglese; in questo corso, per facilitare la comprensione, nomi e commenti ai programmi saranno in lingua italiana, tranne alcuni termini riconosciuti universalmente

Listato 2

```
int LED1      = 6;       // LED1 connesso al pin digitale D6
int LED2      = 5;       // LED2 connesso al pin digitale D5
int Button_1 = 2;       // Button_1 connesso al pin digitale D2
int Button_2 = 7;       // Button_2 connesso al pin digitale D7
int Button_3 = 4;       // Button_3 connesso al pin digitale D4

// Dichiarazione delle variabili usate per leggere lo stato dei pulsanti
int Button_1_State = 0;
int Button_2_State = 0;

void setup()   {
  // inizializza il pin del LED come uscita digitale:
  pinMode(LED1, OUTPUT);
  // initializza il pin del pulsante come ingresso:
  pinMode(Button_1, INPUT);
  // initializza il pin del pulsante come ingresso:
  pinMode(Button_2, INPUT);
}

// Viene dichiarato un loop infinito

void loop() {

  // Legge lo stato del pulsante 1:
  Button_1_State = digitalRead(Button_1);

  // Se è premuto attiva il LED:
  if (Button_1_State == HIGH) {
    // Accendo il LED:
    digitalWrite(LED1, HIGH);
  }

  // Legge lo stato del pulsante 2:
   Button_2_State = digitalRead(Button_2);

  // Se è premuto spegne il LED
  if (Button_2_State == HIGH) {
    // Spengo il LED:
    digitalWrite(LED1, LOW);
  }
}
```

in lingua inglese. Tutti i file saranno anche disponibili per il download direttamente dal sito della rivista a corso terminato, nel qual caso sarà sufficiente copiarli in una cartella in modo da renderli immediatamente utilizzabili. Per completezza riportiamo di seguito alcuni riferimenti al linguaggio di programmazione che, come precedentemente ricordato, deriva dall'ANSI C.

Strutture
- void setup()
- void loop()

Controllo
- if
- if...else
- for
- switch case
- while
- do... while
- break
- continue
- return
- goto

Sintassi
- ; (semicolon)
- {} (curly braces)
- // (single line comment)
- /* */ (multi-line comment)

Operazioni Aritmetiche
- = (assignment)
- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulo)

Comparazione
- == (equal to)
- != (not equal to)
- < (less than)
- > (greater than)
- <= (less than or equal to)
- >= (greater than or equal to)

Operazioni booleane
- && (and)
- || (or)
- ! (not)

Operatori
- ++ (increment) -- (decrement)
- += (compound addition)
- -= (compound subtraction)
- *= (compound multiplication)
- /= (compound division)

Costanti
- HIGH | LOW
- INPUT | OUTPUT
- true | false
- Integer Constants

Tipi di dati
- boolean
- char
- byte
- int
- unsigned int
- long
- unsigned long
- float
- double
- string
- array
- void

Conversioni
- int()
- long()
- float()

Funzioni:

Listato 3

```
/*
 Danger_04
 Nota sonora sul Buzzer.
 Per DangerShield su Arduino Duemilanove
 */

// Buzzer connesso a pin 3
int Buzzer = 3;

void setup() {

  pinMode(Buzzer, OUTPUT);

}

void loop() {

      // Suona una nota sul pin 3 alla frequenza di 1000Hz per 200msec:
      tone(Buzzer, 1000, 200);
      delay(1000);
}
```

Digital I/O
- pinMode(pin, mode)
- digitalWrite(pin, value)
- int digitalRead(pin)

Analog I/O
- int analogRead(pin)
- analogWrite(pin, value) - PWM

Advanced I/O
- shiftOut(dataPin, clockPin, bitOr der, value)
- unsigned long pulseIn(pin, value)

Time
- unsigned long millis()
- delay(ms)
- delayMicroseconds(us)

Math
- min(x, y)
- max(x, y)
- abs(x)
- constrain(x, a, b)
- map(value, fromLow, fromHigh, toLow, toHigh)
- pow(base, exponent)
- sq(x)
- sqrt(x)

Trigonometry
- sin(rad)
- cos(rad)
- tan(rad)

Random Numbers
- randomSeed(seed)
- long random(max)
- long random(min, max)

Serial Communication

Usate per comunicare tra schede arduino, oppure tra schede arduino ed il PC. Vengono usati i pin TX ed RX, facenti capo al modulo.

USART del microcontrollore.
- Serial.begin(speed)
- int Serial.available()
- int Serial.read()
- Serial.flush()
- Serial.print(data)
- Serial.println(data)

UTILIZZO DEI LED (SKETCH DANGER_01)

Torniamo adesso al nostro primo programma per la scheda Danger, nel quale, a differenza del programma Blink, useremo l'uscita 6 invece della 13. Il programma sarà quindi come esposto nel **Listato 1**.
Dopo averlo compilato e trasferito al microcontrollore, vedrete immediatamente lampeggiare il LED. La scheda Arduino Duemilanove supporta il reset automatico, così cliccando semplicemente su UpLoad avrete la compilazione automatica, il trasferimento ed anche il reset e quindi, in circa 4 secondi, il vostro programma sarà già operativo all'interno della Arduino.
Analizziamo il software riga per riga, almeno per questo primo esempio: racchiusi tra /* e */ trovate dei commenti che riportano il nome del programma, la funzione svolta e in quale hardware può funzionare.
Ulteriori commenti su di una riga possono essere scritti iniziandola con //.
Nella prima riga "int ledPin = 6" si dichiara una variabile denominata *ledPin* e associata al pin 6.
La riga di codice racchiusa nella struttura "Void Setup()" specifica che la variabile ledPin associata al pin 6 è utilizzata come uscita. All'interno di questa struttura andremo sempre a specificare la funzione associata per ogni pin utilizzato.
La seconda ed ultima struttura di dati si chiama "Void loop()" e rappresenta semplicemente un loop infinito, ovvero le istruzioni al suo interno vengono eseguite in successione partendo dalla prima dopo la parentesi graffa aperta "{" sino alla fine della struttura delimitata dalla parentesi graffa chiusa "}", per poi essere rieseguite nuovamente all'infinito.
Come vedete, quindi, l'impostazione di un programma risulta assai agevole e semplice.

Tabella 4

Numero visualizzato	Bit inviati	Numero decimale	Segmenti accesi
0	11111100	252	a,b,c,d,e,f
1	01100000	96	a,b
2	11011010	218	a,b,g,e,d
3	11110010	242	a,b,c,d,g
4	01100110	102	b,c,f,g
5	10110110	182	a,f,g,c,d
6	10111110	190	a,c,d,e,f,g
7	11100000	224	a,b,c
8	11111110	254	a,b,c,d,e,f,g
9	11110110	246	a,b,c,d,g,f

USARE I PULSANTI (SKETCH DANGER_02)

L'unità Danger Shield dispone di tre pulsanti; come primo esempio faremo in modo

che premendo Button_1 si accenda il LED1 e premendo Button_2 lo stesso si spenga. Impostiamo quindi il nostro Sketch per poter utilizzare queste due periferiche; il programma corrispondente è visibile nel **Listato 2**.
Le prime righe del listato dichiarano la corrispondenza che c'è tra le nuove periferiche (pulsanti e LED) e i pin del microcontrollore. Seguono due righe per dichiarare due variabili di tipo intero che saranno usate per leggere lo stato degli ingressi. All'interno della struttura di setup vengono usate le istruzioni pinMode(xxx, INPUT) e pinmode(xxx, OUTPUT) per definire quale pin sia di ingresso e quale quello di uscita. Nella struttura del loop viene usata l'istruzione Button_1_State = digitalRead(Button_1); per leggere il livello logico del pin specificato ed inserire questo valore nella corrispondente variabile. Il valore di tale variabile viene usato nell'istruzione successiva if (Button_1_State == HIGH) per testare lo stato del pulsante e, se premuto (livello logico alto) eseguire ulteriori istruzioni, che nel nostro caso coincidono con l'accendere il LED1. Analizzando lo schema elettrico della scheda Danger possiamo infatti vedere che i pulsanti sono connessi ognuno tra il relativo pin digitale e l'alimentazione positiva, mentre una resistenza è connessa tra tali pin e la massa, allo scopo di garantire un livello logico basso (LOW) se il pulsante non è premuto. A pulsante premuto la tensione di alimentazione giungerà al pin corrispondente ponendolo a livello logico alto (HIGH). Le ultime istruzioni del listato testano il pulsante 2 e spengono il LED qualora esso fosse premuto. Le istruzioni contenute all'interno di void loop() vengono ripetute all'infinito per testare in continuazione lo stato dei pulsanti ed accendere e spegnere, di conseguenza, il LED.

UTILIZZO DEL BUZZER (SKETCH DANGER_04)

Vediamo ora come emettere delle semplici note acustiche tramite il cicalino di cui la Danger Shield è equipaggiata. L'istruzione da utilizzare si chiama *tone* e la sintassi di scrittura è la seguente: *tone(pin, frequency, duration)*. Tale istruzione emette un'onda quadra con duty-cycle del 50 % sul pin specificato alla frequenza *frequency*, per la durata *duration* espressa in millisecondi. Tra gli esempi disponibili nella cartella example potrete trovare il file *pitches.h*, che una volta aperto con Word Pad vi fornirà la corrispondenza tra frequenza e note; lo stesso file può essere incluso nell'applicazione. Un esempio di ciò è disponibile in file/example/digital/toneKeyboard. Un ulteriore esempio di emissione

Listato 4

```
/*
 Danger_05
 Visualizza in sequenza i numeri dal 0 al 9 sul display a LED.
 Per DangerShield su Arduino Duemilanove
 */

//Pin connesso a ST_CP di 74HC595
int latchPin = 8;
//Pin connesso a SH_CP di 74HC595
int clockPin = 12;
////Pin connesso a DS di 74HC595
int dataPin = 13;

// Corrispondenza tra i segmenti accesi ed i pin attivi:
int numero[] = {252, 96, 218, 242, 102, 182, 190, 224 , 254, 246};

void setup() {
  //imposta i pin come uscite
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}


void loop() {
  //routine di conteggio
  for (int j = 0; j < 10; j++) {
    //metto bassa la linea latch e predispongo per la trasmissione seriale
    digitalWrite(latchPin, LOW);
    //shift sequenza dei bit in ingresso
    shiftOut(dataPin, clockPin, LSBFIRST, numero[j]);
    //trasferisco il numero sul display
    digitalWrite(latchPin, HIGH);
    delay(1000);  }
}
```

sonora, ma più sofisticato, lo potete trovare in file/example/digital/tonemelody. Per utilizzare questi esempi dovete modificare la riga che specifica la connessione del buzzer (o dell'altoparlante) la quale, nel nostro caso, è la linea 3. L'uso dell'istruzione è semplicissimo, come mostra l'esempio nel **Listato 3**. Il programma si limita ad emettere una nota ogni secondo. L'istruzione delay(1000) crea, semplicemente, un ritardo di 1.000 millisecondi (1 secondo appunto) tra la nota e la successiva.

Listato 5

```
/*
 Danger_06
 Lettura ingresso analogico.
 Viene letta la posizione del Slider1 e visualizzata sul display con valori
 da 0 a 9.
 Per DangerShield su Arduino Duemilanove
 */

//Slider 1 connesso all'ingresso analogico 0
int Slider_1 = 2;
//Variabile che definisce la posizione dello slider
int Slider_Pos = 0;
//Pin connesso a ST_CP di 74HC595
int latchPin = 8;
//Pin connesso a SH_CP di 74HC595
int clockPin = 12;
////Pin connesso a DS di 74HC595
int dataPin = 13;

// Corrispondenza tra i segmenti accesi ed i pin attivi:
int numero[] = {252, 96, 218, 242, 102, 182, 190, 224 , 254, 246};

void setup() {
  //imposta i pin come uscite
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
    // legge il valore del potenziometro
    Slider_Pos = analogRead(Slider_1);
    //Converto il valore letto 0-1023 in un valore 0-9
    Slider_Pos = Slider_Pos/103;
    //metto bassa la linea latch e predispongo per la trasmissione seriale
    digitalWrite(latchPin, LOW);
    //trasferisco in seriale il valore da visualizzare
    shiftOut(dataPin, clockPin, LSBFIRST, numero[Slider_Pos]);
    digitalWrite(latchPin, HIGH);
    delay(200);
}
```

UTILIZZO DEL DISPLAY A LED (SKETCH DANGER_05)

Vediamo adesso come visualizzare un numero sul display in dotazione al modulo, con la premessa che il programma necessario allo scopo è leggermente più complesso di quelli visti finora. Il display a sette segmenti ha bisogno di sette linee di comando, ciascuna delle quali accende un specifico segmento; accendendo i segmenti in modo opportuno siamo in grado di visualizzare un numero decimale tra 0 e 9. In teoria servirebbero sette linee del microcontrollore più un'ottava per il punto decimale, ma è possibile risparmiare I/O, usando opportunamente uno shift-register tipo 74HC595 quale convertitore da seriale a parallelo. Vediamone il funzionamento in dettaglio. Portando la linea Latch a livello logico basso, si bloccano le uscite allo stato attuale; tramite le linee Clock e Data si inviano uno per volta i singoli bit di un numero a 8 bit (chiamato byte) il quale verrà successivamente posto sulle otto uscite non appena la linea latch sarà riportata al livello logico alto. Lo svantaggio di questa procedura è ovviamente il ritardo impiegato per inviare in sequenza tutti gli otto bit, ma sono sufficienti solo tre linee del microcontrollore. Ovviamente dovremo fare in modo che agli otto bit spediti in sequenza corrisponda l'accensione di un determinato numero di segmenti del display tale da far visualizzare una cifra. Per fare questo dovremo crearci una corrispondenza tra il numero inviato ed i segmenti accesi, come indicato nella **Tabella 4**. Il programma che riguarda lo sketch per la gestione del display è illustrato nel **Listato 4**. Oltre alle solite dichiarazioni che ora tralasciamo, vediamo che una riga dichiara un vettore di dieci elementi ciascuno contenente il numero decimale corrispondente ai segmenti da accendere. Non serve far altro che trasferire ogni singolo numero al 595 in modo seriale, cioè bit per bit, e per questo useremo un'apposita funzione, chiamata *shiftout*, che si accolla l'onere di spedire bit per bit il numero richiesto. Prima di richiamare questa funzione dovremmo portare a livello basso la linea del latch per bloccare la visualizzazione sul numero attuale e predisporre l'integrato alla ricezione della sequenza seriale; successivamente porteremo la linea di latch al valore alto per trasferire

Listato 6

```
/*
 Danger_06
 Lettura ingresso analogico.
 Viene letta la posizione del Slider1 e visualizzata sul display con valori da 0 a 9.
 Per DangerShield su Arduino Duemilanove
 */

//Slider 1 connesso all'ingresso analogico 0
int Slider_1 = 2;
//Variabile che definisce la posizione dello slider
int Slider_Pos = 0;
//Pin connesso a ST_CP di 74HC595
int latchPin = 8;
//Pin connesso a SH_CP di 74HC595
int clockPin = 12;
////Pin connesso a DS di 74HC595
int dataPin = 13;

// Corrispondenza tra i segmenti accesi ed i pin attivi:
int numero[] = {252, 96, 218, 242, 102, 182, 190, 224 , 254, 246};

void setup() {
  //imposta i pin come uscite
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
}

void loop() {
    // legge il valore del potenziometro
    Slider_Pos = analogRead(Slider_1);
    //Converto il valore letto 0-1023 in un valore 0-9
    Slider_Pos = Slider_Pos/103;
    //metto bassa la linea latch e predispongo per la trasmissione seriale
    digitalWrite(latchPin, LOW);
    //trasferisco in seriale il valore da visualizzare
    shiftOut(dataPin, clockPin, LSBFIRST, numero[Slider_Pos]);
    digitalWrite(latchPin, HIGH);
    delay(200);
}
```

il numero alle uscite dell'integrato, così da aggiornare il numero visualizzato sul display. Queste funzioni sono racchiuse all'interno di un ciclo "for" che le ripete per ciascun numero dallo zero al nove con un ritardo di un secondo tra un numero e l'altro.

USO DEGLI INGRESSI ANALOGICI (SKETCH DANGER_06)

Vediamo ora come poter utilizzare gli ingressi analogici, ai quali sono connessi i potenziometri, i sensori di luce e temperatura ed il microfono. L'istruzione che andremo ad utilizzare si chiama *AnalogRead* e permette, appunto, di leggere il livello di tensione all'ingresso di un pin analogico. Specifichiamo subito che mentre un ingresso digitale può assumere solo due valori, che sono uno o zero (ad esempio pulsante premuto oppure rilasciato), un ingresso analogico accetta tutte le tensioni comprese tra lo zero ed il potenziale di alimentazione (in questo caso 5 volt). La scheda Arduino dispone di 6 canali analogici con convertitore analogico digitale a 10 bit, che però non possono essere letti direttamente dal microcontrollore, il quale riconosce solo componenti digitali. A trasformare i segnali analogici in dati digitali riconoscibili dal micro, provvede il modulo interno denominato ADC (analog to digital converter); la risoluzione di 10 bit significa che il numero associato alla lettura analogica potrà andare da un minimo di zero fino ad un massimo di 1.023 (2^{10}-1). Essendo la corrispondenza lineare, possiamo dire che se a 5 volt corrisponde il numero 1.023, a 2,5 corrisponde 511 e via di seguito. Va da sè che la risoluzione sarà al massimo 5/1024 ovvero circa 4,9 millivolt. Il tempo necessario alla conversione è di circa 100 microsecondi e quindi la massima frequenza di campionamento potrebbe essere di 10 kHz, ovvero diecimila campioni al secondo. La sintassi con cui scrivere l'istruzione è: *analogRead(pin)*, dove *pin* indica il piedino del micrcontrollore dal quale sarà eseguita la lettura. La funzione ritorna il valore della

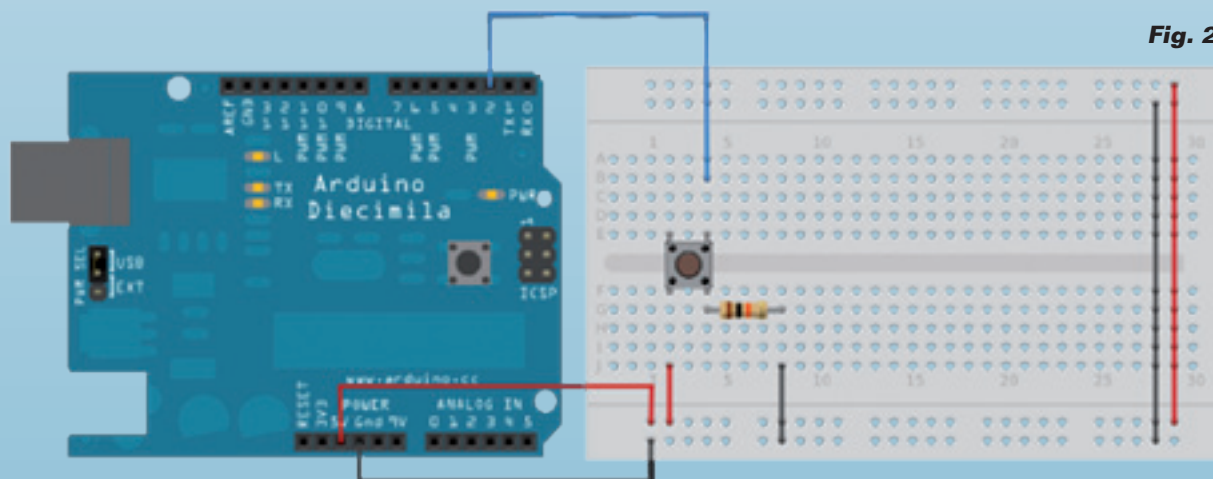



Fig. 2

conversione, che, come detto prima, sarà compreso tra 0 e 1.023.
Come al solito, prepariamo un esempio per dimostrare l'utilizzo della nostra istruzione: creiamo uno sketch che permetta di visualizzare sul display la posizione dello slider_1 tramite un numero tra 0 e 9. Ovviamente non è possibile leggere direttamente il valore del potenziometro, ma come avete visto dallo schema, i due capi del potenziometro sono collegati uno a massa ed uno alla tensione di alimentazione, mentre il terzo contatto, quello centrale, è connesso all'ingresso del microcontrollore. In questo modo abbiamo creato un partitore resistivo che permette di far giungere all'ingresso solo una parte della tensione di alimentazione a seconda della posizione dello stick dello slider.
Il programma che permette di visualizzare il numero corrispondente alla posizione del cursore del potenziometro è esposto nel **Listato 5**. Abbiamo utilizzato il listato del precedente esercizio per consentire la visualizzazione sul display, però aggiungendo semplicemente due righe necessarie alla lettura dell' ingresso analogico.
La riga *int Slider_1 = 2;* dichiara la lettura analogica dall'ingresso 2, al quale è connesso lo slider_1, mentre la *int Slider_Pos = 0;* definisce una variabile che useremo per scalare il valore numerico da 1023 a 9 per adattarlo alla capacità del display. Segue la riga che avvia la conversione, ossia *Slider_Pos = analogRead(Slider_1);*, la quale salva il risultato nella variabile *Slider_Pos* e successivamente lo scala riconducendolo ad un valore compreso tra 0 e 9. Ciò viene ottenuto semplicemente dividendo la variabile per 103:
Slider_Pos = Slider_Pos/103;. Il valore così ottenuto viene visualizzato sul display con le istruzioni precedentemente descritte.
Se appena alimentata l'unità noterete dei valori strani sul display, premete il pulsante di reset e tutto tornerà a posto.

Fig. 3

USARE LA FOTORESISTENZA (SKETCH DANGER_07)

In modo del tutto simile è possibile visualizzare sul display l'intensità luminosa che colpisce la fotoresistenza: è sufficiente modificare il canale di lettura specificando quello cui è connessa la fotoresistenza. Anche in questo caso per avere una tensione che dipenda dal livello luminoso, è sufficiente creare un partitore di tensione con un'altra resistenza (R7) come visibile nello schema. Applicando la legge di Ohm possiamo anche determinare la tensione che giunge al microcontrollore, secondo la formula:

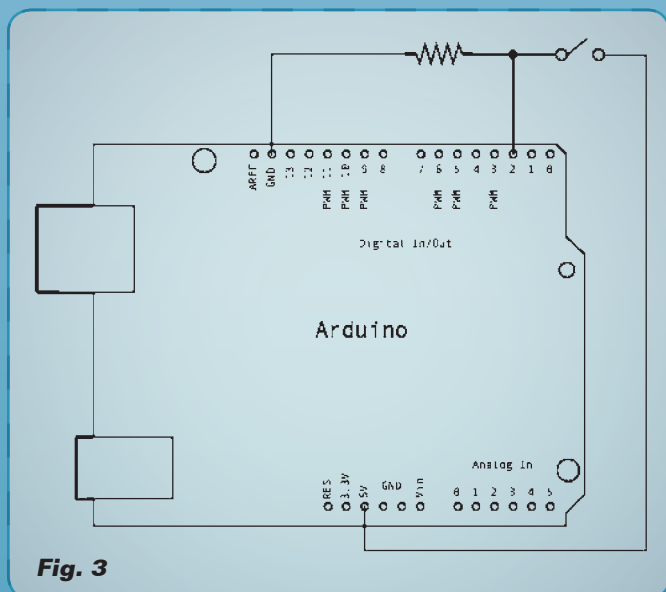
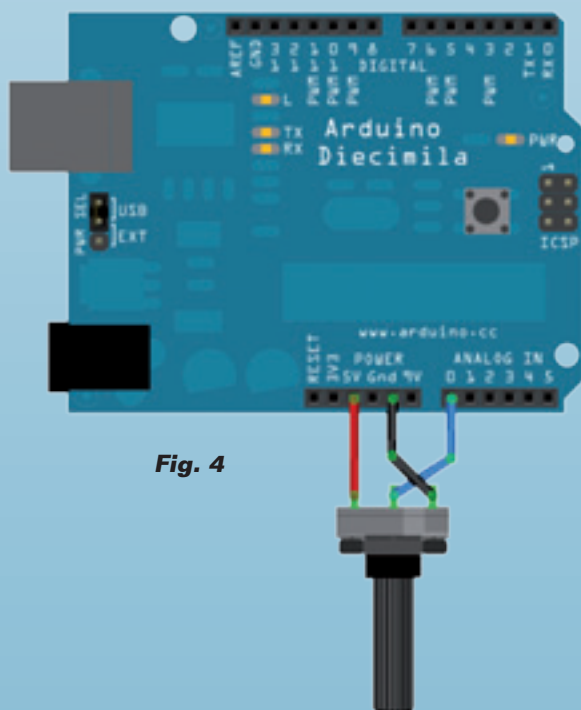



Fig. 4

$$Vin=5 \cdot R7/(R7+R_{LDR})$$

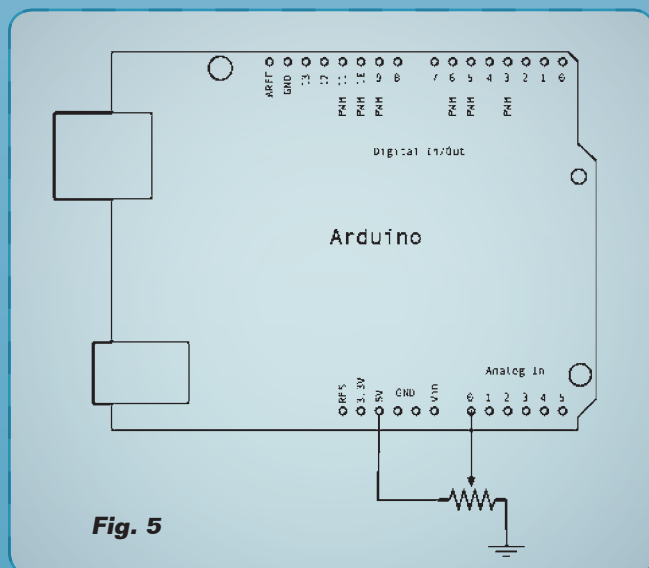
Tale tensione dipende dal valore resistivo della fotoresistenza, il quale a sua volta è funzione dell'intensità luminosa.

UTILIZZO DEL SENSORE DI TEMPERATURA (SKETCH DANGER_08)

Anche in questo caso nulla di nuovo: il canale analogico da leggere è il "4" ed il programma è facilmente adattabile. Il sensore utilizzato, un LM35, fornisce 10 mV per ogni grado centigrado rilevato e quindi per una temperatura ambiente di 20 °C erogherà una tensione di soli 200 mV, a dire il vero un po' pochini per poter realizzare una misura ben fatta. Ipotizzando una temperatura massima di circa 40 °C, raggiungibile scaldando con le mani il sensore, possiamo anche stimare il massimo valore numerico fornito dal convertitore ADC=1.023·(0,4/5)=82. In virtù di ciò, possiamo quindi tarare la lettura affinché con il valore di 40 °C sul display venga visualizzato il massimo valore, cioè 9.

Fig. 5

USO DEL SENSORE DI COLPI (SKETCH DANGER_09)

Il sensore di colpi è lo stesso buzzer usato per generare le note acustiche: sollecitato meccanicamente, la capsula piezoelettrica che si trova al suo interno genera una piccola tensione che può essere rilevata tramite l'ingresso analogico del microcontrollore. Possiamo quindi realizzare un semplice programma che possa captare i colpi sul sensore per accendere momentaneamente un LED; lo trovate nel **Listato 6**. Chi non disponesse della Danger Shield potrà comunque ripetere tutte le esperienze qui presentate, a patto di connettere alla MainBoard Arduino le giuste periferiche. Così, ad esempio, se volessimo collegare un pulsante sull'ingresso digitale 2 dovremmo fare come descritto nelle figure 2 e 3, visibili in queste pagine. Si tratta di connettere gli stessi componenti usati della Danger Shied, ma direttamente alla mainboard; in questo caso ci siamo aiutati con una breadboard esterna per facilitare i collegamenti con dei semplici spezzoni di filo. Se volessimo collegare un potenziometro all'ingresso analogico 0 potremmo fare riferimento alle figure 4 e 5. Un'altra alternativa è l'utilizzo della PROTOSHIELD, che similmente alla Danger board, è una scheda aggiuntiva da inserire nel connettore della MainBoard, creandone di fatto un'estensione. A differenza della Danger, non sono già connesse le varie periferiche come i pulsanti i LED ecc., però è presente un'area con delle piazzole alle quali sarete voi a decidere cosa collegare e quali collegamenti con il microcontrollore fare. Tale soluzione offre senz'altro più flessibilità di quella permessa dalla Danger board, in quanto potete anche modificare a vostro piacimento le periferiche togliendole e inserendole al bisogno.
Bene, con questa lezione è tutto; vi aspettiamo alla prossima puntata.

Conoscere e usare Arduino

dell'ing. MIRCO SEGATELLO

Scopriamo come gestire display a cristalli liquidi alfanumerici e grafici, vedendo alcuni sketch di esempio validi per i più comuni modelli reperibili in commercio.

Siamo giunti ormai alla terza puntata del corso su Arduino; questa volta l'argomento della lezione sono i display: non solo quelli di testo ma anche quelli grafici. Faremo una carrellata di applicazioni con diversi tipi di dispositivi, sempre in modo molto semplice e proponendo esempi pratici che riguardino la parte sia hardware che software. I primi dispositivi che andremo a descrivere sono i display di testo, immancabili in svariate applicazioni e molto semplici da utilizzare, grazie alle librerie già pronte. Prima di addentrarci nei dettagli, facciamo una breve introduzione su questo tipo di display, non tanto dal punto di vista costruttivo, quanto riguardo al loro utilizzo. Partiamo dicendo che tutti i display LCD di testo si basano su un chip decoder, che, letti i dati in ingresso, provvede a gestire i pixel del display in modo appropriato. È proprio il tipo di decoder usato all'interno del display che ne determina l'utilizzo e le funzioni, non tanto la grandezza o il numero di righe; è quindi importante, quando si acquista un display LCD, sapere quale chip decoder lo governa, perché da questo dipenderà la sua gestione a livello software. A livello hardware,

Tabella 1

+5V	Alimentazione positiva
GND	Massa
VLCD	Tensione negativa usata per la regolazione del contrasto
Data	8 linee di dati
WR/RD	Linea per la lettura/scrittura per LCD
E	Comando di abilitazione display
RS	Linea per impostazione dati/comandi

invece, le cose sono più complicate, perché ogni casa produttrice, anche a parità di chip decoder montato, personalizza la piedinatura dei contatti, rendendo a volte inutilizzabili display privi di documentazione. Per questo corso useremo esclusivamente display LCD basati sul controller Hitachi HD44780 oppure Samsung ST7066U: si tratta di display con interfaccia parallela per i quali i più svariati ambienti di sviluppo per microcontrollori includono già i comandi per controllarli, il che risparmia la fatica di dover elaborare specifiche routine di controllo. L'interfaccia hardware è standard e prevede i contatti raggruppati nella **Tabella 1**.
Oltre a queste linee, potrebbero essere disponibili i contatti della retroilluminazione (BL+ e BL-) da utilizzarsi se è richiesta la visione in ambienti poco illuminati.
Ovviamente i nomi possono cambiare da produttore a produttore, così come il riferimento al pin del connettore al quale fanno capo, però la funzione rimane sempre la stessa.
Facciamo ora un esempio pratico, utilizzando un display basato su chip Hitachi che presenta il connettore posto in alto e la piedinatura riportata nella **Tabella 2**.

In questo display la retroilluminazione (in inglese *backlight*) viene attivata semplicemente ponendo BL- a massa e BL+ al positivo dei 5 volt con una resistenza da 20÷50 ohm a seconda del livello di intensità luminosa desiderato. A questo punto siamo pronti al cablaggio fisico del display con la scheda Arduino; per fare questo ci sono diverse possibilità, ad esempio l'utilizzo della Protoshield oppure di una basetta sperimentale. L'importante è rispettare lo schema di collegamento illustrato nella **Fig. 1**. Facciamo ora un esempio pratico utilizzando il display di codice CDL4162 (Clover) distribuito dalla ditta Futura Elettronica, dal cui sito è possibile scaricare anche il data-sheet in lingua italiana, nel quale è riportata chiaramente la piedinatura, con nomi e funzioni di ogni singolo pin.
È inoltre riportata la mappa dei caratteri visualizzabili, che si basa su un codice ASCII rivisto per contenere numeri, lettere e simboli in diverse lingue.
La corrispondenza tra pin e funzioni è riportata nella **Tabella 3**.
Per la retroilluminazione potete fare riferimento alle indicazioni precedenti. Per il cablaggio dovete fare riferimento allo schema elettrico illustrato nella **Fig. 2**.
In ogni caso è necessario effettuare i seguenti collegamenti tra la scheda Arduino e il display LCD:
• LCD RS → pin digitale 12;

Tabella 2

Pin	Segnale	Funzione
1	VSS	Alimentazione (0V)
2	VDD	Alimentazione (5V)
3	VO	Drive LCD (0V rispetto VDD)
4	RS	Alto) ingresso codici di istruzione Basso) ingresso dati
5	R/W	(Alto) lettura dati (Basso) scrittura dati
6	E	Segnale di abilitazione
7	DB0	Linea di bus dati
8	DB1	
9	DB2	
10	DB3	
11	DB4	
12	DB5	
13	DB6	
14	DB7	
15	BL-	Terminale di alimentazione LED (-)
16	BL+	Terminale di alimentazione LED (+)

Tabella 3

Pin	Segnale	Funzione
1	BL+	Terminale di alimentazione LED (+)
2	BL-	Terminale di alimentazione LED (-)
3	GND	Alimentazione (0V)
4	VDD	Alimentazione (5V)
5	VO	Drive LCD
6	RS	Alto) ingresso codici di istruzione Basso) ingresso dati
7	R/W	(Alto) lettura dati (Basso) scrittura dati
8	E	Segnale di abilitazione
9	DB0	Linea di bus dati
10	DB1	
11	DB2	
12	DB3	
13	DB4	
14	DB5	
15	DB6	
16	DB7	

- LCD R/W → GND;
- LCD Enable → pin digitale 11;
- LCD D4 → pin digitale 5;
- LCD D5 → pin digitale 4;
- LCD D6 → pin digitale 3;
- LCD D7 → pin digitale 2.

Oltre a ciò bisogna provvedere a collegare l'alimentazione ed il trimmer per la regolazione del contrasto. I pin D0, D1, D2, D3 possono essere lasciati liberi oppure posti a massa.
A questo punto avete il vostro display cablato col microcontrollore, ma non ancora programmato; se lo accendete, non noterete nulla se non dei pixel parzialmente attivi.
Al momento non risulta tarato neppure il contrasto.

Fig. 1

Per completezza riportiamo qui di seguito anche tutti i comandi disponibili in fase di programmazione e gli esempi disponibili nell'ambiente di sviluppo.

Function
- LiquidCrystal()
- begin()
- clear()
- home()
- setCursor()
- write()
- print()
- cursor()
- noCursor()
- blink()
- noBlink()
- display()
- noDisplay()
- scrollDisplayLeft()
- scrollDisplayRight()
- autoscroll()
- noAutoscroll()
- leftToRight()
- rightToLeft()
- createChar()

Examples
- Hello World
- Blink
- Cursor
- Display
- Text Direction
- Autoscroll
- Serial input

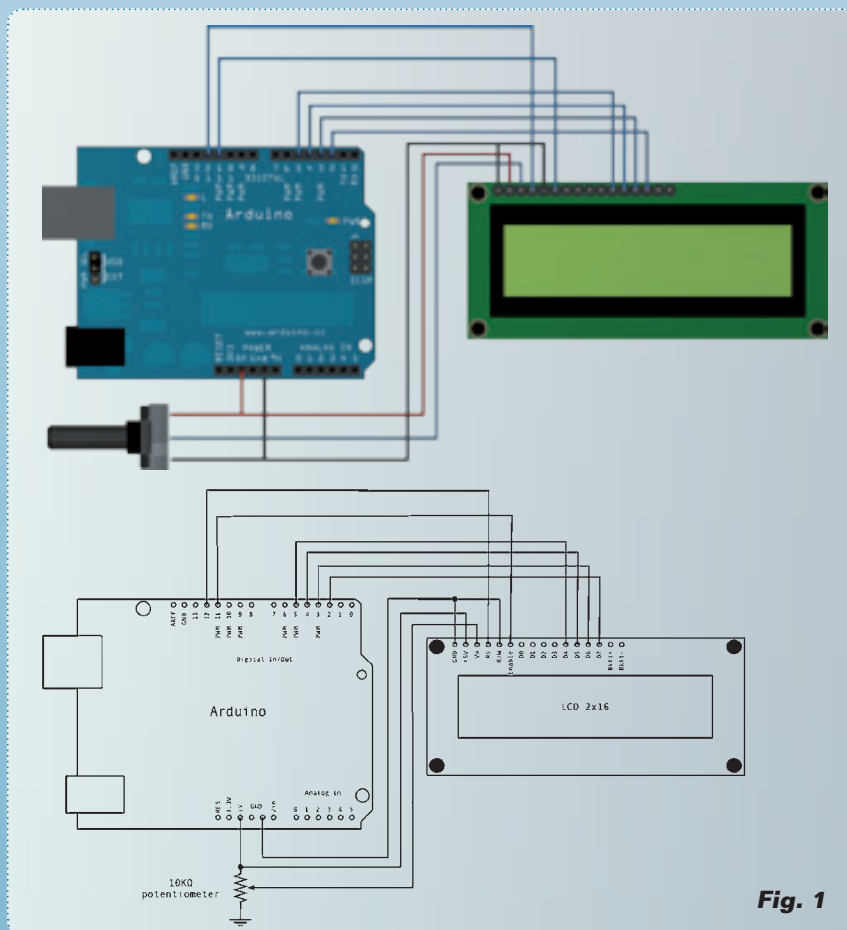
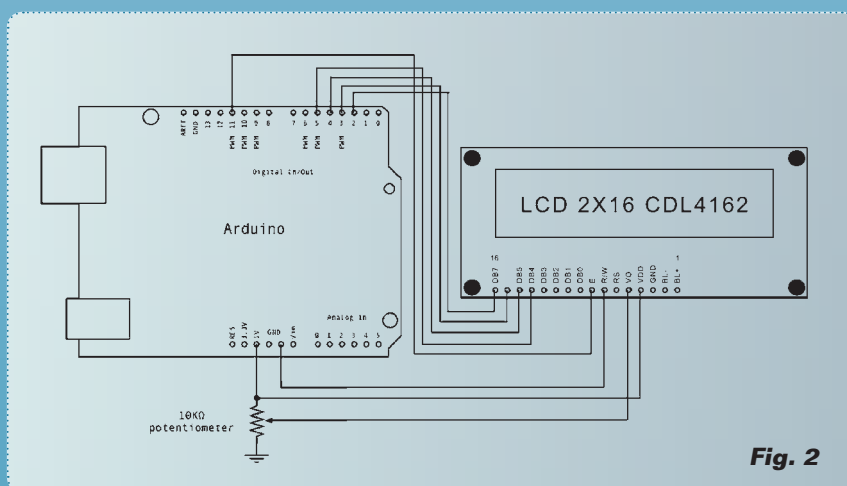



Fig. 2

Listato 1

```
/*
   Display_01

   Semplice scritta su diaply LCD di testo.
   Per il cablaggio hardware:

 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2

 */
// include la libreria LCD:
#include <LiquidCrystal.h>

// initializza il display con l'interfaccia
// hardware specificata
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup() {
  // setta il numero di colonne e righe del display:
  lcd.begin(16, 2);
  // Scrive un messaggio.
  lcd.print("hello, world!");
}

void loop() {
  // Imposta il cursore all'inizio della seconda riga
  // (nota: il conteggio delle righe parte da zero:
  lcd.setCursor(0, 1);
  // Scrive il numero di secondi a partire
  // dall'istante di reset:
  lcd.print(millis()/1000);
}
```

• SetCursor
• Scroll

Adesso apriamo il software di sviluppo Arduino (la versione 18, nel nostro caso), creiamo un nuovo sketch oppure, più velocemente, utilizziamo un esempio già pronto disponibile nella cartella del programma *File/examples/liquidcrystal/helloworld*.
Sul nostro sito, a fine corso, saranno disponibili anche i programmi commentati in lingua italiana usati nelle varie puntate.
Lo sketch cui facciamo riferimento è descritto nel **Listato 1**.

Caricate ed avviate lo sketch, regolate il contrasto agendo sul trimmer sino ad evidenziare la scritta "hello, world!" ed il gioco è fatto. Questo esempio è molto importante perché vi fa capire anche la modalità con la quale vengono visualizzate le variabili, in questo caso quella denominata *millis()* , che contiene il numero di millisecondi da quando Arduino sta eseguendo il programma; dividendola per 1.000 si ottiene il numero di secondi di esecuzione del programma.
Notate come la variabile scritta in memoria in forma binaria venga rappresentata in numero decimale cifra per cifra, con una conversione automatica. Altro punto importante da notare è la presenza della riga #include <LiquidCrystal.h> che permette di includere la libreria e quindi i comandi per gestire un display LCD. Allo stesso modo sarà possibile importare altre librerie, anche create da terzi, al fine di espandere le funzioni disponibili.
Potete sbizzarrirvi e provare anche tutti gli altri esempi inerenti ai display LCD: ci sono esempi per agire sulla scritta oppure sul cursore in vari modi. Per comodità riportiamo, nella **Tabella 4** di seguito, la sintassi con un esempio dei principali comandi per il display.
Passiamo ora all'utilizzo dei display LCD grafici (GLCD) sicuramente più affascinanti e flessibili rispetto a quelli di testo; cercheremo di non addentrarci troppo nello specifico, rimandando, chi volesse approfondire il discorso sul controllo dei display grafici, all'apposito corso già pubblicato nei fascicoli dal n° 115 al n° 123.
La prima cosa da dire è che come per i display LCD di testo, i GLCD si suddividono a seconda del chip adibito al controllo e sono essenzialmente disponibili sul mercato due grandi categorie: la prima si basa sull'uso

Tabella 4 *- Comandi per il display e loro sintassi.*

comando	funzione
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);	Inizializza il display e lo dichiara come oggetto di nome lcd da usarsi nel programma
lcd.begin(cols, rows);	Imposta per l'oggetto lcd il numero di colonne (cols) ed il numero di righe(rows)
lcd.cursor();	Attiva visualizzazione del cursore sulla posizione attuale
lcd.noCursor();	Disattiva visualizzazione del cursore sulla posizione attuale
lcd.blink();	Attiva il lampeggio del cursore
lcd.noBlink();	Disattiva il lampeggio del cursore
lcd.display()	Attiva visualizzazione sul display
lcd.noDisplay();	Disattiva visualizzazione sul display
lcd.scrollDisplayRight();	Trasla il testo del display di una posizione a destra
lcd.scrollDisplayLeft();	Descrizione: Trasla il testo del display di una posizione a sinistra

del chip KS0107B che però consente una sola risoluzione fissa pari a 128x64. Questo tipo di display è abbastanza comune ed economico, ma la risoluzione non troppo elevata lo rende poco flessibile. Una seconda categoria si basa sull'uso del cip T6963C, il quale permette di gestire display con le seguenti risoluzioni: 128x128, 240x64 e 240x128.
Esistono in commercio svariati altri modelli di GLCD, ma non sempre sono disponibili le librerie per il loro controllo; quindi, di fatto risultano più difficilmente gestibili.
Come primo esempio prendiamo in considerazione il display modello LM12864MBC distribuito dalla Futura Elettronica (codice 1446-LCD102B6B) e basato sul chip KS0107B, quindi con risoluzione 128x64 in bianco e nero. Come già spiegato per le prove con il display di testo, è necessario realizzare il cablaggio elettrico e poi programmare adeguatamente la scheda Arduino; i collegamenti sono quelli descritti nella **Tabella 5**.
Visto il carattere sperimentale dell'applicazione, anche in questo caso utilizziamo una breadbord su cui eseguire i collegamenti con dei fili rigidi. È possibile saldare sul display un connettore strip femmina come quelli presenti sulla scheda Arduino e da lì, sempre con dei filetti rigidi, eseguire i collegamenti. Sarebbe anche possibile saldare direttamente dei fili sul display oppure utilizzare la Protoshield per eseguire tutti i collegamenti con dei filetti saldati. Ad ogni modo, dovete essere molto diligenti e seguire in modo scrupoloso la **Tabella 5**, ricontrollando più volte il lavoro eseguito. Il trimmer, del valore di 10÷20 kohm, ha tre terminali: i due esterni vanno connessi uno a massa ed uno al pin 18 del display (Vout) mentre il terminale centrale

Tabella 5 *- Collegamenti fisici tra il display LM12864MBC e Arduino Duemilanove.*

Pin Arduino	Pin display	Nome	funzione
GND	1	VSS	GND
+5V	2	VDD	+5V
Pin2 (centrale) del trimmer	3	Vo	Contrasto LCD
17 (analog3)	4	RS	Data/instruction
16 (analog2)	5	R/W	Read/Write
18 (analog4)	6	E	Enable
8	7	DB0	Dato
9	8	DB1	Dato
10	9	DB2	Dato
11	10	DB3	Dato
4	11	DB4	Dato
5	12	DB5	Dato
6	13	DB6	Dato
7	14	DB7	Dato
14 (analog0)	15	CS1	Chip select 1
15 (analog1)	16	CS2	Chip select 2
Reset (porre a +5V)	17	RST	Reset
Pin1 del trimmer Pin3 del trimmer a GND	18	Vout	Tensione per Vo
+5V (se retro.zione ON)	19	BLA	Retroilluminazione anodo
GND	20	BLK	Retroilluminazione catodo

Tabella 9 *- Collegamenti fisici tra il display ADM12864H e Arduino Duemilanove.*

Pin Arduino	Pin display	Nome	funzione
GND	2	VSS	GND
+5V	1	VDD	+5V
Pin2 (centrale) del trimmer	3	Vo	Contrasto LCD
17 (analog3)	16	D/I	Data/instruction
16 (analog2)	15	R/W	Read/Write
18 (analog4)	17	E	Enable
8	4	DB0	Dato
9	5	DB1	Dato
10	6	DB2	Dato
11	7	DB3	Dato
4	8	DB4	Dato
5	9	DB5	Dato
6	10	DB6	Dato
7	11	DB7	Dato
14 (analog0)	13	CS1	Chip select 1
15 (analog1)	12	CS2	Chip select 2
Reset (porre a +5V)	14	RST	Reset
Pin1 del trimmer / Pin3 del trimmer a GND	18	Vout	Tensione per Vo
+5V (se retro.zione ON)	19	BLA	Retroilluminazione anodo
GND	20	BLK	Retroilluminazione catodo

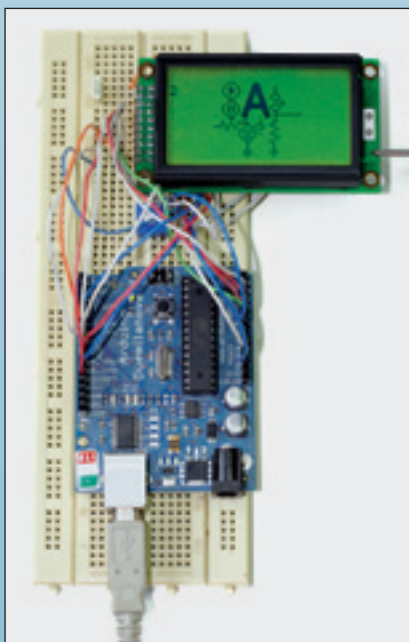



Fig. 3

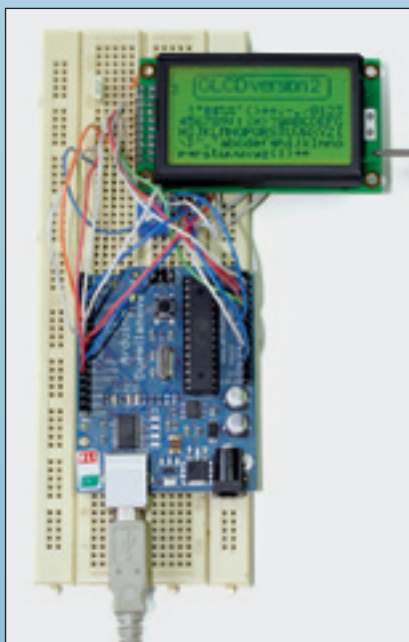



Fig. 4

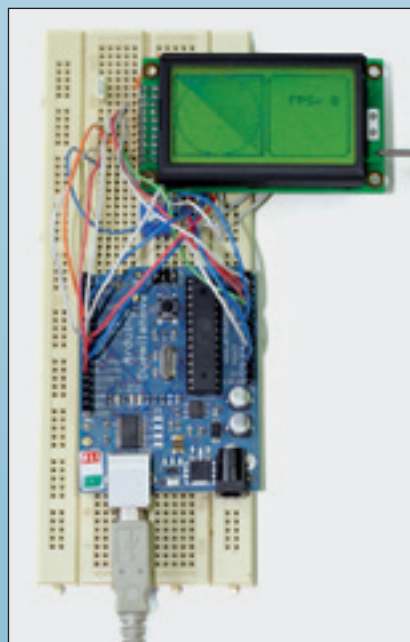



Fig. 5

va collegato al pin 3 (Vo) sempre del display. Non guasta inserire un piccolo condensatore da 100 nF tra i +5 V e il GND, il più vicino possibile al display per sopprimere eventuali disturbi elettrici presenti nei collegamenti. Fatto questo, si passa alla programmazione della scheda Arduino, per la quale dobbiamo procurarci le librerie specifiche per il display grafico LCD con chip K0108, sviluppate in questo caso da terze parti. Sempre sul sito ufficiale in lingua inglese di Arduino, potete trovare una sezione dedicata a "KS0108

comando	descrizione
GLCD.Init(NON_INVERTED)	Inizializza la libreria per il disegno normale o invertito. Se normale ogni pixel viene disegnato (appare scuro), se invertito il pixel viene cancellato.
GLCD.GotoXY(x,y)	Posiziona il cursore nella posizione x e y, dove 0,0 è l'angolo superiore sinistro.
GLCD.ClearScreen()	Cancella lo schermo.
Graphic Drawing Functions: se color è WHITE cancella pixel; se BLACK attiva pixel.	
GLCD.DrawCircle(x, y, radius, color)	Disegna un cerchio alle coordinate x,y di raggio radius.
GLCD.DrawLine(x1,y1,x2,y2,color)	Disegna una linea dalle corrdinate x1,y1 alle x2,y2.
GLCD.DrawVertLine(x, y, length, color)	Disegna una linea verticale dalle coordinate x,y di lunghezza length.
GLCD.DrawHoriLine(x, y, length, color)	Disegna una linea orizzontale dalle cordinate x,y di lunghezza length.
GLCD.DrawRect(x, y, width, height, color)	Disegna un rettangolo.
GLCD.DrawRoundRect(x, y, width, height, radius, color)	Disegna un rettangolo con gli angoli smussati.
GLCD.FillRect(x, y, width, height, color)	Disegna un rettangolo pieno.
GLCD.InvertRect(x, y, width, height)	Inverte i pixel del rettangolo specificato.
GLCD.SetInverted(invert)	Imposta il modo di disegno invertito.
GLCD.SetDot(x, y, color)	Disegna un pixel alla posizione x,y.
GLCD.DrawBitmap(bitmap, x, y, color)	Disegna un'immagine bitmap alle coordinate x,y.
Font Functions	
GLCD.SelectFont(font, color)	Imposta i font dei caratteri.
GLCD.PutChar(character)	Scrive un carattere.
GLCD.Puts(string)	Scrive una stringa.
GLCD.Puts_P(string)	Scrive una stringa contenuta nella memoria del programma.
GLCD.PrintNumber(number)	Scrive il valore decimale di una variabile numerica.
GLCD.CursorTo(x, y)	Imposta le coordinate di base per i font a larghezza fissa.

***Tabella 6** - Comandi per la gestione dei display grafici, ripartiti per categoria: Graphic Drawing Functions riguardano il disegno di primitive grafiche e Font Functions è inerente ai caratteri.*

Tabella 7 - Collegamenti fisici tra scheda Arduino e Graphic LCD serial backpack LCD-09352.

Pin Arduino	Pin LCD-09352	
Vin	Vin	Alimentazione positiva 6-7V
GND	GND	GND
TX	RX	Linea dati

Graphics LCD library" che spiega in dettaglio l'utilizzo di questa libreria e ne permette anche il download. I nostri lettori troveranno già disponibili questi file, arricchiti da un ulteriore esempio; la cartella di nome KS0108GLCD contiene sia le libreria, sia dei file di esempio e deve essere copiata all'interno della cartella *Libraries sottodirectory* della cartella principale *Arduino-0018*.
Aprite lo sketch di nome *GLCDexample.pde*, che troverete nel percorso *File→example→Ks0108* e caricatelo sulla scheda Arduino. Una volta avviato il firmware, compariranno tre schermate in sequenza, la prima con un'immagine, la seconda con tutti i caratteri stampati e la terza con linee e cerchi, come mostrato nelle figure, rispettivamente, 3, 4, 5. Per vedere bene le rappresentazioni potrebbe essere necessario regolare il contrasto del display agendo sul trimmer.
Lo sketch appena descritto dimostra l'utilizzo della libreria GLCD per KS0108 visualizzando a video sia immagini (bitmap in questo caso) sia testo e grafica. Se utilizzate i file scaricati dal nostro sito Internet, troverete anche un ulteriore esempio denominato *GLCDelettroni-*

Tabella 8

funzione	Bytes [esadecimale]	Note
Cancellazione	0x7C + 0x00	
Demo mode	0x7C + 0x04	
Reverse Mode	0x7C + 0x12	
Set retroilluminazione	0x7C + 0x02 + R	R=valore illuminazione 0-100
Set/reset pixel	0x7C + 0x10 + X + Y + P	X=coordinata orizzontale Y=coordinata verticale P=0 resetta P=1 setta
Disegna Linea	0x7C + 0x0C + X1 + Y1 + X2 + Y2 + P	X1,Y1=coordinate di inizio X2,Y2=coordinate di fine P=0 cancella P=1 disegna
Disegna cerchio	0x7C + 0x03 + X1 + Y1 + R + P	X1,Y1=coordinate centro R=raggio P=0 cancella P=1 disegna
Disegno rettangolo	0x7C + 0x0F + X1 + Y1 + X2 + Y2 + P	X1,Y1=angolo alto sinistro X2,Y2=angolo basso destro P=0 cancella P=1 disegna
Modifica baudrate	0x7C + 0x07 + N	N="1" = 4800bps N="2" = 9600bps N="3" = 19,200bps N="4" = 38,400bps N="5" = 57,600bps N="6" = 115,200bps
Set coordinate	0x7C + 0x18 + X	X=nuova coordinata X
Set coordinate	0x7C + 0x19 + Y	Y=nuova coordinata Y

Listato 2

```
/*********************************************************/
/*  Configuration for assigning LCD bits to Arduino Pins */
/*********************************************************/
/* Arduino pins used for Commands
 * default assignment uses the first five analog pins
 */

#define CSEL1                          14              // CS1 Bit   // swap pin assignments with CSEL2 if
left/right image is reversed
#define CSEL2                          15              // CS2 Bit
#define R_W                                    16              // R/W Bit
#define D_I                                    17              // D/I Bit
#define EN                                     18              // EN Bit
//#define RES                                     19           // Reset Bit  // uncomment this to contol
LCD reset on this pin

/* option: uncomment the next line if all command pins are on the same port for slight speed & code size im-
provement */
#define LCD_CMD_PORT               PORTC    // Command Output Register for pins 14-19
/* Arduino pins used for LCD Data
 * un-comment ONE of the following pin options that corresponds to the wiring of data bits 0-3
 */
#define dataPins8to11 // bits 0-3 assigned to arduino pins 8-11, bits 4-7 assigned to arduino pins 4-7
//#define dataPins14to17 //bits 0-3 assigned to arduino pins 14-17, bits 4-7 assigned to arduino pins 4-7.
(note command pins must be changed)
//#define dataPins0to3  // bits 0-3 assigned to arduino pins 0-3 , bits 4-7 assigned to arduino pins 4-7, this
is marginally  the fastest option but  its only available on runtime board without hardware rs232.

/* NOTE: all above options assume LCD data bits 4-7 are connected to arduino pins 4-7 */

/*******************************************************/
/*     end of Arduino configuration                                        */
/*******************************************************/
```

cain, che permette di visualizzare a video una semplice scritta. Lo sketch semplificato è più semplice da comprendere rispetto all'esempio originale. Un secondo esempio denominato *GLCDeinline* disegna due linee incrociate a tutto schermo, con al centro un cerchio.
La libreria è molto nutrita e l'elenco completo di tutti i comandi è riportato, con le descrizioni del caso, nella **Tabella 6**.
Come avete visto, il display grafico richiede 8 linee per i dati e 5 per il controllo; l'assegnazione di queste funzioni è esplicitata nel file *KS0108_Arduino.h* con le righe di programma visibili nel **Listato 2**. È possibile modificare queste linee di codice per impostare in modo diverso le linee utilizzate per la gestione del display GLCD; a riguardo, si consiglia prima di leggere le note esplicative riportate nella pagina HTML di riferimento per questa libreria, nel sito di Arduino. Utilizzando un microcontrollore a 28 pin rimangono comunque poche linee disponibili per le applicazioni. Una seconda possibilità è offerta dal display grafico ADM12864H, anche questo reperibile presso la Futura Elettronica (codice LCD128x64) sempre basato sul chip KS0108 che presenta il vantaggio di avere i 20 pin in linea e quindi facilmente gestibili con una breadbord. Per il collegamento alla scheda Arduino potete fare riferimento alla **Tabella 9**.
Una valida soluzione alle difficoltà di cablaggio ed al cospicuo numero di linee utilizzato, arriva con l'introduzione sul mercato dei display a controllo seriale, i quali necessitano di sole quattro linee (+Vcc, GND, TX, RX) che si comandano tramite semplici stringhe inviate in modalità seriale. In commercio sono anche

Listato 3

```
/*
   Display_02

   Semplice esempio utilizzo GLCD a comando seriale.
   Per il cablaggio hardware:
   Graphic LCD Serial Backpack connesso a GLCD-ADM12864H
   Usare le linee Vin, GND, TX.
 */

byte buf_erase[]  = {0x7C, 0x00};                   // Cancella schermo
byte buf_backl[]  = {0x7C, 0x02, 0};                // (retroilluminazione 0=0% 100=100%)
byte buf_line[]   = {0x7C, 0x0C, 0, 0, 127, 63, 1}; // Linea (X1,Y1) (X2,Y2) 1=disegna 0=cancella
byte buf_circle[] = {0x7C, 0x03, 63, 31, 20, 1};    // Cerchio (X1,Y1) raggio 1=disegna 0=cancella


void setup()
{
  Serial.begin(115200);          // Imposta comunicazione a 115200 baud
}

void loop()                      // Ripete all'infinito
{
  Serial.write(buf_backl, 3);    // Retroilluminazione 0%
  delay(100);
  Serial.write(buf_erase, 2);    // Cancella schermo
  delay(1000);
  Serial.write(buf_line, 7);     // Disegna linea
  delay(1000);
  Serial.write(buf_circle, 6);   // Disegna cerchio
  delay(1000);
  Serial.print("ElettronicaIN"); // Scrive un testo
  delay(1000);
}
```

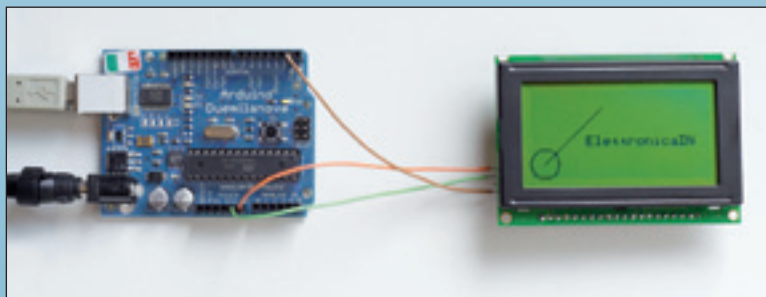
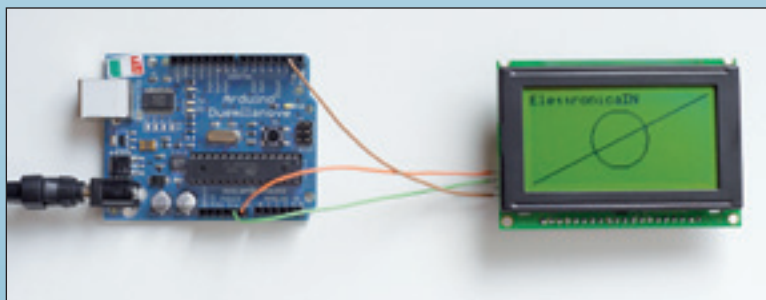
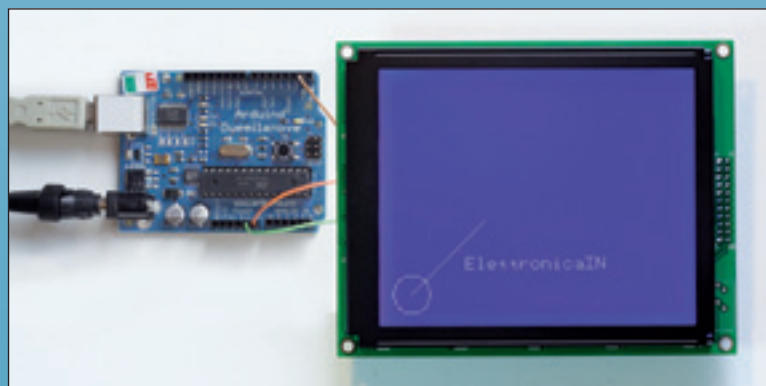
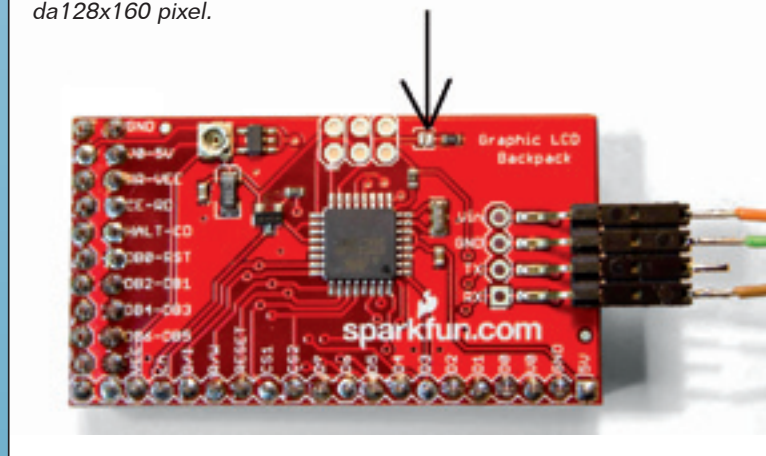
Listato 4

```
/*
   Display_03

   Semplice esempio utilizzo GLCD a comando seriale.
   Si usano i comandi a riga
   Per il cablaggio hardware:
   Graphic LCD Serial Backpack connesso a GLCD-ADM12864H
   Usare le linee Vin, GND, TX.
 */

void setup()
{
  Serial.begin(115200);      // imposta comunicazione a 115200 baud
}

void loop()                  // ripete all'infinito
{
  Serial.print(0x7C, BYTE);  // Cancella schermo
  Serial.print(0x00, BYTE);
  delay(100);
  Serial.print(0x7C, BYTE);  // Retroilluminazione 0%
  Serial.print(0x02, BYTE);
  Serial.print(0x00, BYTE);
  delay(100);
  Serial.print(0x7C, BYTE);  // Disegna un cerchio
  Serial.print(0x03, BYTE);  //
  Serial.print(100, BYTE);   // Coordinata X
  Serial.print(50, BYTE);    // Coordinata Y
  Serial.print(10, BYTE);    // Raggio
  Serial.print(1, BYTE);     // 1=disegna 0=cancella
  delay(1000);
  Serial.print(0x7C, BYTE);  // Set coordinata X
  Serial.print(0x18, BYTE);  // Coordinata X
  Serial.print(10, BYTE);    // Coordinata X
  Serial.print(0x7C, BYTE);  // Set coordinata Y
  Serial.print(0x19, BYTE);  // Coordinata Y
  Serial.print(40, BYTE);    // Coordinata Y
  Serial.print("ElettronicaIN");
  delay(1000);
}
```

disponibili dei convertitori seriale/parallelo per display GLCD, denominati Graphic LCD serial backpack: un esempio è il modello LCD-09352, perfettamente compatibile con il display grafico ADM12864H (foto nella pagina precedente in alto a sinistra). In questo caso il cablaggio è veramente minimo, perché sono sufficienti le tre linee descritte nella **Tabella 7**. La scheda aggiuntiva va innestata sul connettore del display; nel nostro caso abbiamo usato uno strip maschio sul display ed uno femmina sull'adattatore. Siccome la scheda aggiuntiva internamente ricava i 5 volt tramite un proprio stabilizzatore, va alimentata con una tensione leggermente più alta 6÷7 V (raccomandati); il massimo ammesso è 9 volt. Ecco perché la scheda Arduino dovrà venire alimentata tramite un alimentatore esterno, dato che la tensione della USB non è sufficiente. Per i nostri esperimenti abbiamo utilizzato un semplice alimentatore non stabilizzato da 0,5 A, impostato per ottenere una tensione di uscita di 6 volt; anche se può sembrare poco, in realtà con tale impostazione in questo genere di alimentatori è facile prelevare anche 8÷9 volt. Sul pin Vin della scheda Arduino è presente la tensione appena uscita dall'alimentazione e non ancora stabilizzata a 5 volt; dovremo avere l'accortezza di fornire in ingresso non più di uno o due volt oltre detto valore. Le gestione di questo adattatore avviene semplicemente con l'invio di comandi seriali che la scheda Arduino gestisce in fase di programmazione tramite le due linee (TX ed RX) che sono le stesse utilizzate per la comunicazione con la USB (tramite il convertitore FT232). Riassumiamo, nella **Tabella 8**, i principali comandi da inviare al display, che devono essere composti secondo lo standard RS232, impostando come parametri di comunicazione 115200,N,8,1 ovvero 115.200 baud (valore predefinito ma modificabile), nessun bit di parità, 8 bit di dati ed un bit di stop.
Nel **Listato 3** trovate il programma usato per testare il display, che permette di impostare la retroilluminazione a zero (spenta) cancella

Listato 5

```
/*
   Display_04

   Semplice esempio utilizzo GLCD a comando seriale.
   Si usano SubRoutine per i comandi
   Per il cablaggio hardware:
   Graphic LCD Serial Backpack connesso a GLCD-ADM12864H
   Usare le linee Vin, GND, TX.
 */

void setup()
{
  Serial.begin(115200);      // imposta comunicazione a 115200 baud
}

void loop()
{
  backlight(0);                    // Retroilluminazione a 0%
  clearLCD();                      // Cancella schermo
  cursorSet(40,30);                // Imposta posizione per il testo
  Serial.print("ElettronicaIN");   // scrive un testo nella posizione indicata
  line(10,10,50,50);               // Disegna una linea
  circle(10,10,10);                // Disegna un cerchio
  delay(2000);
}

//  SubRoutine

// cancella LCD
void clearLCD(){
  Serial.print(0x7C, BYTE);
  Serial.print(0x00, BYTE);
}

// gestione retroilluminazione
void backlight(byte light){
  Serial.print(0x7C, BYTE);
  Serial.print(0x02, BYTE);
  Serial.print(light); // imposter zero per disabilitare la retroilluminazione
}

// disegna una linea
void line(byte x1, byte y1, byte x2, byte y2){
  byte buf_line[]   = {0x7C, 0x0C, 0, 0, 127, 63, 1}; // Linea (X1,Y1)
  Serial.print(0x7C, BYTE);  // Disegna un cerchio
  Serial.print(0x0C, BYTE);
  Serial.print(x1, BYTE);    // Coordinata X1
  Serial.print(y1, BYTE);    // Coordinata Y1
  Serial.print(x2, BYTE);    // Coordinata X2
  Serial.print(y2, BYTE);    // Coordinata Y2
  Serial.print(1, BYTE);      // 1=disegna 0=cancella
}

// Disegna un cerchio
void circle(byte xpos, byte ypos, byte radius){
  Serial.print(0x7C, BYTE);     // Disegna un cerchio
  Serial.print(0x03, BYTE);     //
  Serial.print(xpos, BYTE);     // Coordinata X
  Serial.print(ypos, BYTE);     // Coordinata Y
  Serial.print(radius, BYTE);   // Raggio
  Serial.print(1, BYTE);        // 1=disegna 0=cancella
}

// muove il cursore alla posizione X, Y specificata
void cursorSet(byte xpos, byte ypos){
  Serial.print(0x7C, BYTE);  // Set coordinata X
  Serial.print(0x18, BYTE);  // Coordinata X
  Serial.print(xpos, BYTE);    // Coordinata X
  Serial.print(0x7C, BYTE);  // Set coordinata Y
  Serial.print(0x19, BYTE);  // Coordinata Y
  Serial.print(ypos, BYTE);    // Coordinata Y
}
```

lo schermo e disegna una linea, un cerchio e una scritta. Come vedete, le scritte sono immediate grazie al generatore di caratteri incluso nel backpack; è infatti sufficiente inviare il codice ASCII relativo al carattere da stampare a video. Ovviamente non dovrete fare alcuna conversione ma solo usare l'istruzione *Serial.print* seguita dalla stringa da stampare racchiusa tra virgolette. Un'alternativa all'invio dei comandi appena descritti è riportata nel **Listato 4**, in cui sono vedete una porzione di programma che spedisce i byte necessari uno alla volta. Vi è anche una terza alternativa, che consiste nel costruirsi delle subroutine da richiamare ogni qualvolta si voglia fare una funzione; questa soluzione la trovate meglio descritta nel **Listato 5**. Se avete modo di scaricare il data-sheet di questo backpack, noterete che risulta compatibile anche con un altro display "Huge Graphic LCD 160x128 pixel" di grandi dimensioni e con risoluzione maggiore. La gestione a livello software avviene nello stesso modo in quanto, ovviamente, la scheda Arduino dialoga ancora con il backpack, che riesce a gestire il display in completa autonomia. Con la retroilluminazione spenta la visibilità è ridotta, quindi abbiamo previsto un apposito scketch (display_05) che imposta anche una minima retroilluminazione. Il Backpack viene normalmente fornito compatibile con il display 128x64, ma è possibile impostarlo per funzionare con il display 128x160: allo scopo basta rimuovere il jumper posto sullo stampato. Tale jumper è davvero minuscolo ed è realizzato con una piccola saldatura (una goccia di stagno) che dovrà essere rimossa con molta attenzione. Bene, anche per questa puntata è tutto; avete tempo fino alla prossima per fare tutti gli esperimenti e le prove pratiche descritte.

Le due piazzole da cui rimuovere il cortocircuito a stagno per impostare la scheda Arduino in modo da farle controllare un display da128x160 pixel.

Conoscere e usare Arduino

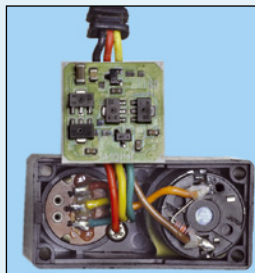
dell'ing.
MIRCO
SEGATELLO

Impariamo a gestire gli attuatori ed in particolare vediamo come comandare servocomandi da modellismo, stepper-motor e motori a spazzole, tramite semplici routine firmware.

Siamo giunti alla quarta puntata del corso dedicato al sistema di sviluppo per microcontrollori Arduino. Stavolta ci occupiamo di attuatori, ovvero di tutti quei componenti che convertono energia elettrica in movimento; nello specifico, parleremo di servocomandi da modellismo, motori a spazzole e motori passo/passo. I servo da modellismo, come sapete, sono utilizzati appunto in campo modellistico per poter muovere alcune parti di un modello, come ad esempio lo sterzo in un'automobilina oppure il piano di coda di un aereo. Il segnale di comando giunge dalla ricevente di bordo e non è inusuale trovare modelli con quattro o più servocomandi dedicati ad altrettanti movimenti.
Utilizzare un servo da modellismo con la scheda Arduino è davvero molto semplice: basta semplicemente cablare tre fili ed avviare il programma corrispondente.
Per la programmazione dobbiamo fare riferimento alla libreria standard *Servo.h*, che andiamo a descrivere di seguito e che si compone delle istruzioni seguenti.

Il servo da modellismo

Nati per impiego modellistico, si trovano molto diffusi oggigiorno anche in applicazioni elettroniche, grazie alla loro versatilità e facilità di gestione in tutte quelle situazioni in cui è necessario eseguire un movimento meccanico di precisione. Dal costo contenuto e forniti in svariate grandezze, dipendenti essenzialmente dalla loro potenza, possono essere impiegati in tantissime applicazioni pratiche: una tra tutte, la movimentazione di piccole videocamere per le riprese a bordo di robot (pan/tilt).

Per comprendere il funzionamento di un servo da modellismo, occorre osservarlo al suo interno: esso è costituito da un piccolo motore in corrente continua che, grazie ad un sistema di ingranaggi, fa ruotare un perno sul quale è calettato un piccolo potenziometro; la lettura del valore resistivo di questo potenziometro fornisce la posizione esatta del perno. Un circuito elettronico realizzato in tecnologia SMD provvede al controllo bidirezionale del motore ed al corretto posizionamento del perno in relazione al segnale elettrico di comando.

Il tutto funziona secondo lo schema riportato nella **Fig. 1**, dal quale appare evidente che il posizionamento avviene confrontando il valore in tensione fornito dal potenziometro con quello ricavato dal segnale di ingresso e ruotando di conseguenza il motore sino a quando questi due valori non coincidono perfettamente.

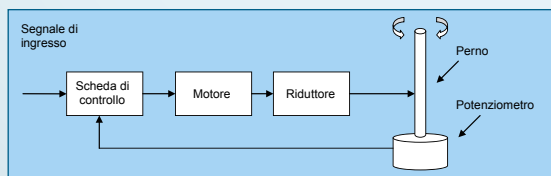



Fig. 1

In questo modo si ottiene un controllo di posizione molto veloce e preciso, comandabile con semplici segnali elettrici. Il cavetto di collegamento è composto da un filo di riferimento (GND), un filo per l'alimentazione (da 4,8 a 6 volt) ed un filo per il segnale di comando (**Tabella 1**).

colore	funzione
Nero o marrone	Negativo di alimentazione (GND)
rosso	Positivo di alimentazione (+Vcc)
Giallo o bianco	Segnale di comando (ingresso)

Tabella 1

Occorre dire che non è prevista la rotazione continua (salvo casi particolari) del perno, ma solo di ±60° rispetto alla posizione iniziale, anche se è possibile espandere la rotazione sino a ±90 °. Il segnale di controllo è di tipo PWM (Pulse Wide Modulation) formato da impulsi ad onda rettangolare ripetuti ogni 20 ms, la cui "larghezza" permette di impostare la posizione del perno del servo. La posizione centrale si ottiene quando gli impulsi hanno una durata di 1,5 ms (**Fig. 2**).

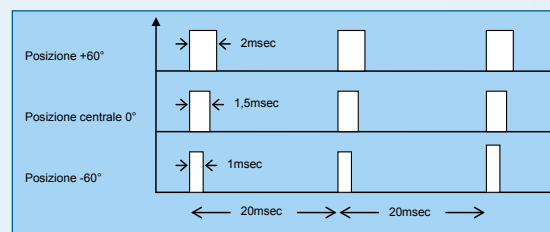



Fig. 2 *- Il segnale di comando.*

Questo tipo di segnale digitale si presta benissimo ad essere generato da una logica programmabile, quindi i servo possono essere comodamente gestiti dai microcontrollori.

ISTRUZIONE ATTACH()

Associa la variabile servo ad uno specifico pin. In Arduino 0016 e precedenti, sono supportati solo due servo collegati ai pin 9 and 10.
Sintassi: *servo.attach(pin)*, *servo.attach(pin, min, max)*.
Parametri servo: variabile di tipo Servo.
Pin: numero del pin harware utilizzato.
Min (opzionale): durata minima dell'impulso, in microsecondi, corrispondente al minimo grado di rotazione (0 gradi) del servo (il valore predefinito è 544).
Max (opzionale): durata massima dell'impulso, in microsecondi, corrispondente alla massima rotazione (180 gradi) del servo (il valore predefinito è 2400).

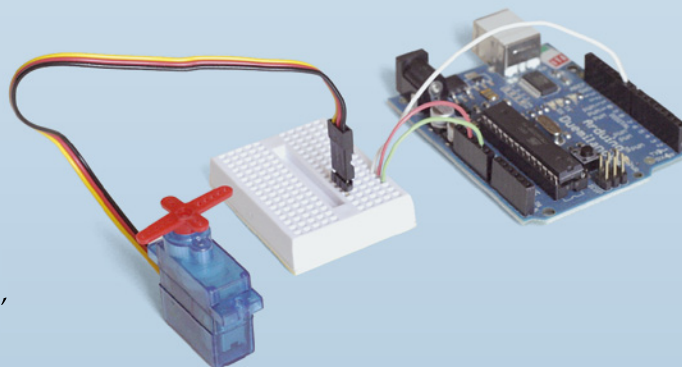
ISTRUZIONE ATTACHED()

Verifica l'associazione tra la variabile servo ed il pin.
Sintassi: *servo.attached()*.
Parametri servo: variable di tipo Servo.

Ritorno: vero se il servo è associato al pin; falso in caso contrario.

ISTRUZIONE DETACH()
Dissocia la variabile servo al pin specificato. Se tutte le variabili servo non sono associate, i pin 9 e 10 possono essere usati come uscite PWM con l'istruzione *analogWrite()*.
Sintassi: *servo.detach()*.
Parametri: servo= variabile di tipo servo.

ISTRUZIONE READ()
Legge l'attuale posizione del servo corrispondente all'ultima posizione passata con l'istruzione *write()*.
Sintassi: *servo.read()*.
Parametri servo: variabile di tipo servo.
Ritorno: l'angolo del servo da 0 a 180 gradi.

ISTRUZIONE WRITE()
Invia il valore in gradi relativo alla posizione del perno del servo. Un valore 0 corrisponde alla massima rotazione a sinistra, mentre 180 equivale alla massima rotazione a destra; il valore 90 indica la posizione centrale. L'esatta corrispondenza tra valore in gradi inviato e l'effettiva rotazione del servo viene specificata dai valori *Max* e *Min* nella dichiarazione dell'istruzione *attach()*; tali valori devono essere ricavati mediante prove pratiche, in quanto possono anche variare da servo a servo.
Sintassi: *servo.write(angle)*.
Parametri servo: variabile di tipo servo.
Angle: valore corrispondente alla rotazione in gradi.

ISTRUZIONE WRITEMICROSECONDS()
Imposta la posizione del servo come valore relativo alla durata dell'impulso espressa in microsecondi. Normalmente un valore 1000 corrisponde alla massima rotazione a sinistra, 2000 alla massima rotazione a destra ed il valore 1500 corrisponde alla posizione centrale (neutro)
Sintassi: *servo.writeMicroseconds(µS)*.
Parametri servo: variabile di tipo servo.
µS: valore in microsecondi relativo alla posizione del servo.
Installando il software Arduino-18, vi ritroverete con due esempi già pronti relativi all'utilizzo dei servo: il primo, denominato Knob, permette di posizionare il servo a seconda

Listato 1

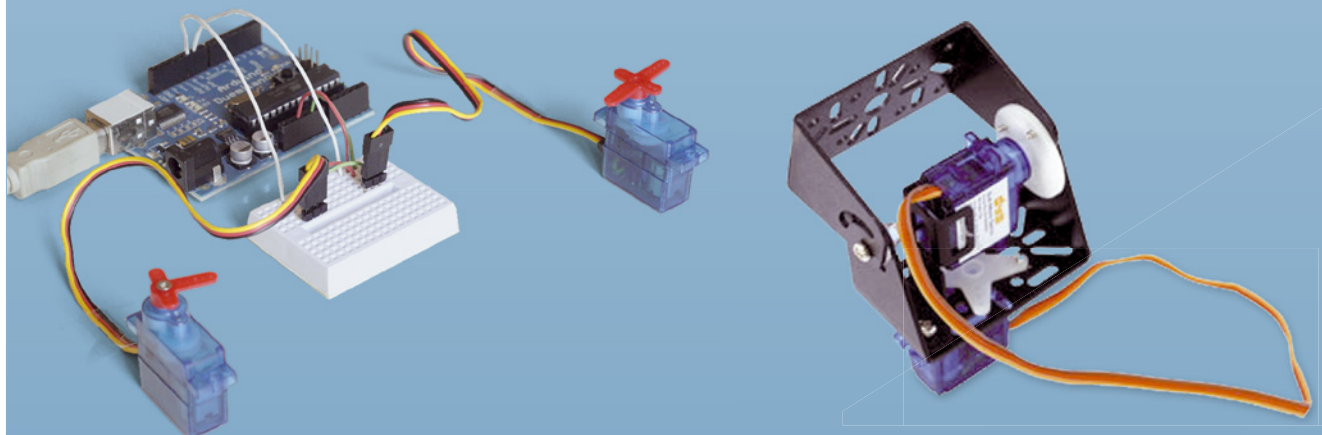
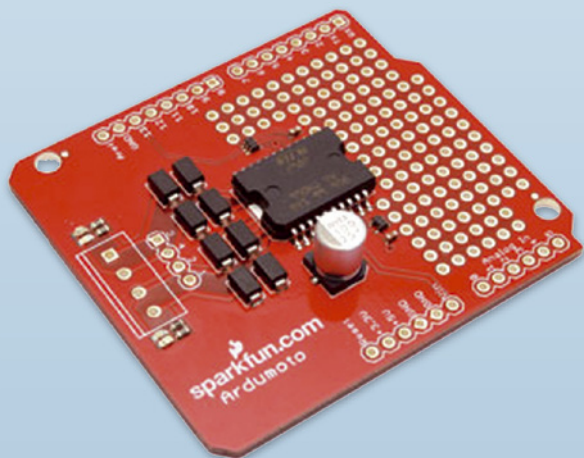
```
#include <Servo.h>

Servo myservo;  // crea un oggetto di tipo servo con nome myservo

int pos = 0;    // variabile contenente il valore della posizione del servo

void setup()
{
  myservo.attach(9);  // associa l'oggetto myservo  al pin 9
}


void loop()
{
  for(pos = 0; pos < 180; pos += 1)                // loop partendo da 0 fino a 180 gradi
  {                                                         // a passi di un grado
    myservo.write(pos);                             // imposta la posizione del servo
    delay(15);                                              // attende che il servo raggiunga la posizione
  }                                 // dal minimo al massimo sono necessari 15msX180=2,7secondi
  for(pos = 180; pos>=1; pos-=1)     // loop da 180 fino a zero gradi
  {
    myservo.write(pos);              // imposta la posizione del servo
    delay(15);                       // attende che il servo abbia raggiunto la posizione
  }
}
```

Listato 2

```
#include <Servo.h>

Servo myservo;  // crea un oggetto di tipo servo con nome myservo

void setup()
{
  Serial.begin(9600);      // imposta comunicazione a 115200 baud
  Serial.println("Pronto!");
}

void loop() {

  static int v = 0;

  if ( Serial.available()) {
    char ch = Serial.read();

    switch(ch) {
      case '0'...'9':
        v = (ch - '0')*20;  // '0'=0° '9'=180°
        myservo.write(v);
        break;
      case 'd':
        myservo.detach();
        break;
      case 'a':
        myservo.attach(9);
        break;
    }
  }
}
```

della posizione di trimmer cablati sulla scheda Arduino, mentre il secondo, denominato Sweep, permette di far girare l'alberino del servo alternativamente dalla posizione minima a quella massima. In entrambi gli sketch il servo è cablato connettendo il positivo al pin +5 V di Arduino, la massa al pin GND e l'ingresso di comando al pin 9. Solo per il primo esempio è necessario collegare anche un trimmer con i contatti esterni connessi uno a +5 V e l'altro a GND ed il centrale (cursore) al pin 0 di Arduino.
Nel **Listato 1** riportiamo il codice relativo al secondo esempio e ne descriviamo il funzionamento, allo scopo di comprendere meglio l'utilizzo di questa libreria.
Come si vede da tale listato, la libreria semplifica notevolmente il lavoro di programmazione; le istruzioni chiave sono quelle che definiscono un oggetto di tipo servo *myservo=Servo* che successivamente sarà associato ad uno specifico pin *myservo.attach(9)*.
Fatto questo, per impostare la posizione del servo è sufficiente utilizzare il comando *myservo.write(pos)* con il parametro *pos* che può valere tra 0 e 180 (corrispondenti ad una rotazione tra 0° e 180°).
Possiamo ora realizzare uno sketch che ci permetta di impostare la posizione di un servo direttamente da PC, inviando la posizione tramite il tool Serial-Monitor.
Il programma per questa funzione è descritto dal **Listato 2**.
Il listato, peraltro molto semplice, prevede di inizializzare un servo "myservo" ed associarlo al pin 9; viene quindi abilitata la comunicazione seriale con il comando *Serial.begin(9600)*.
La riga di codice *char ch = Serial.read()* permette di attendere l'arrivo di un carattere e di salvarlo nella variabile *ch* che successivamente, con l'istruzione *case*, viene usata per eseguire le istruzioni di associazione servo ('a'), dissociazione servo ('d') oppure posizionamento del servo (numeri da '0' a '9').
Per provare questo programma lasciate il servo connesso al pin 9 ed avviate *Serial monitor* dal menu *Tools*; assicuratevi di aver impostato

Arduino	Motorshield
Pin 12	Controllo direzione motore A
Pin 10	Segnale PWM per controllo velocità motore A
Pin 13	Controllo direzione motore B
Pin 11	Segnale PWM per controllo velocità motore B
	Morsetti 1 e 2 collegamento motore A
	Morsetti 3 e 4 collegamento motore B

Tabella 2

una velocità di comunicazione di 9.600 baud. Aspettate che si evidenzi la scritta "pronto!" inviata dalla scheda Arduino appena terminata la programmazione, quindi spedite il carattere "a" per abilitare il servo; successivamente digitate un numero tra 0 e 9 ed inviatelo (pulsante *send*). Il servo viene posizionato tra 0° e 180° in passi di 20°.
La gestione di due servocomandi è altrettanto facile, essendo sufficiente dichiarare due oggetti di tipo servo ad esempio *Servo_1* e *Servo_2*, associarli alle uscite 9 e 10 e comandarli con le istruzioni *Servo_1.write(pos1)* e *Servo_2.write(pos2)*. La gestione di un sistema pan/tilt per il puntamento di una videocamera risulta molto semplice; un esempio di sketch lo troverete assieme ai sorgenti di questa puntata, con il nome di *motor_2*. Per la parte meccanica consigliamo di utilizzare due servocomandi (codice SERVO206) in abbinamento al pan/tilt bracket kit di codice PANTILTKIT, il tutto reperibile presso la ditta Futura Elettronica (*www.futurashop.it*).
Vediamo ora come sia possibile gestire dei motori a spazzole in corrente continua, di quelli, per intenderci, che normalmente vengono usati nei giocattoli e che spesso troviamo in molte applicazioni di robotica. Per questo è disponibile uno specifico hardware denominato ArduMoto (la versione V12 nel nostro caso) disponibile presso la Futura Elettronica (il codice del prodotto è 7300-MOTORSHIELD).
Questa scheda viene fornita già montata con componenti in SMD, è basata sul chip L298 e permette di controllare direzione e velocità di 2 motori DC con una corrente massima di 2 ampere ciascuno. Alimentata direttamente dalla linea Vin di Arduino Duemilanove o Seeeduino, ogni sua uscita dispone di un LED blu e uno giallo per indicare la direzione di rotazione del motore. Tutte le linee di uscita del chip L298 sono protette da un diodo.
Con questa scheda è possibile gestire ciascun

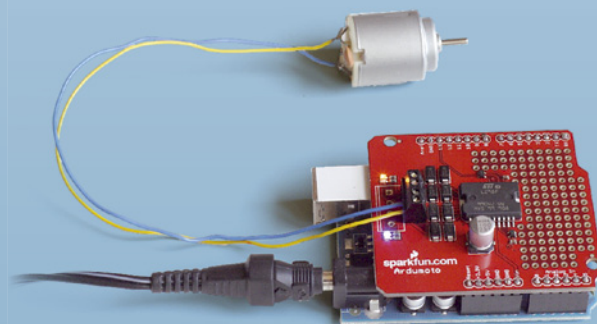



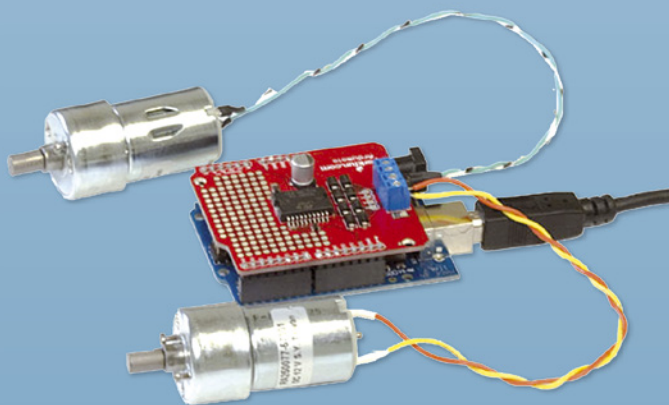
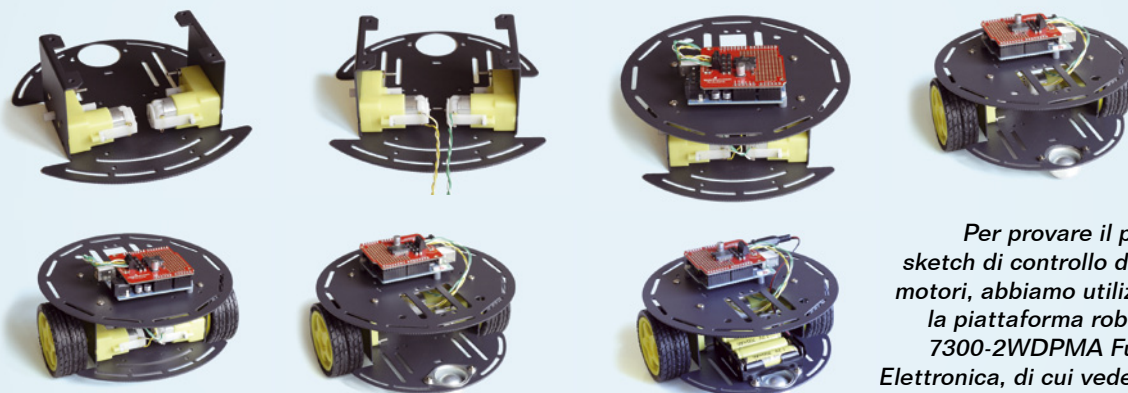
Fig. 3 *– Cablaggio del motore a 6 V.*

Rapporto di riduzione	1:120
Giri a vuoto(3V)	100 RPM
Giri a vuoto(6V)	200 RPM
Corrente a vuoto(3V)	60 mA
Corrente a vuoto(6V)	71 mA
Corrente a rotore bloccato(3V)	260 mA
Corrente a rotore bloccato(6V)	470 mA
Coppia (3V)	1,2 kgcm
Coppia (6V)	1,92 kgcm
Dimensioni	55 x 48,3 x 23 mm
Peso	45 g

Tabella 3

Per provare il primo sketch di controllo di due motori, abbiamo utilizzato la piattaforma robotica 7300-2WDPMA Futura Elettronica, di cui vedete le fasi di montaggio.

motore in entrambi i sensi di marcia, mentre la velocità di rotazione viene regolata con la tecnica del PWM. Ricordiamo in breve che, con il termine PWM, si intende una tecnica di modulazione in cui il segnale in uscita, in questo caso la tensione al motore, viene applicata e poi tolta ad intervalli regolari e molto velocemente. Maggiore è il tempo in cui è presente la tensione in uscita, rispetto al tempo in cui è assente, più il motore girerà velocemente e viceversa.
La ArduMoto si installa direttamente al di sopra della scheda Arduino, con un cablaggio predefinito e riepilogato nella **Tabella 2**.
Allo scopo è necessario che vi procuriate degli strip maschio passo 2,54 ed una morsettiera per contatti passo 2,54 oppure 3,5 mm per il collegamento dei motori; la loro saldatura non presenta alcuna difficoltà.
Per la scelta dei motori, da poter collegare, dovete considerare che l'alimentazione viene prelevata dalla linea Vin della scheda Arduino, che a sua volta coincide con l'alimentazione esterna applica al Plug di alimentazione e può avere un valore compreso tra 7 e 14 V. Per quanto riguarda l'assorbimento dei motori, ri-

pin	con	funzione
Motor A	Jp3	Avvolgimento A.
Motor A	Jp3	Avvolgimento A.
Motor B	Jp3	Avvolgimento B.
Motor B	Jp3	Avvolgimento B.
GND	Jp1	Massa alimentazione motori.
M+	Jp1	Alimentazione positiva motori (8-30volt).
GND	Jp4	Massa integrato A3967.
+5V	Jp4	Alimentazione integrato A3967 (per impostazione di fabbrica, è ricavata dalla tensione motori tramite stabilizzatore interno).
Cur Adj		Imposta la corrente di riposo dei motori (150÷750 mA).
APWR	Sj1	Ponticello (normalmente chiuso) alimentazione integrato dalla tensione motori. Tagliare per alimentare l'integrato dal connettore JP4.
3/5V	Sj2	Ponticello per impostare alimentazione integrato su 5 o 3,3 V e di conseguenza i livelli logici dei segnali di comando. Per impostazione predefinita, normalmente aperto Vcc=5 V e i segnali di comando sono 0÷5 V.
SLP	Jp5	Pone in standby l'integrato. Di default la linea è alta e l'integrato è attivo.
MS1	Jp5	Impostazioni funzione passo: MS1=1 e MS2=1 1/8 di passo (predefinito); MS1=0 e MS2=1 1/4 di passo; MS1=1 e MS2=0 1/2 di passo; MS1=0 e MS2=0 passo intero.
MS2	Jp6	
ENABLE	Jp6	Abilitazione integrato. Per impostazione predefinita, la linea è a livello basso e l'integrato è abilitato.
RST	Jp7	Reset dell'integrato. Per impostazione predefinita la linea è a livello alto e l'integrato non è resettato.
PFD	Jp7	Imposta la rapidità di variazione di corrente tra un passo ed il successivo. Permette di ottimizzare la funzionalità ad alto numero di giri. Preimpostato su di un valore intermedio.
GND	Jp2	Massa segnale di comando.
STEP	Jp2	Impulso di comando corrispondente ad uno step.
DIR	Jp2	Imposta la direzione di rotazione. La direzione di rotazione dipende anche dal cablaggio degli avvolgimenti.

Tabella 4 *– Collegamenti della EasyDriver 4.3.*

I motori passo-passo

I motori passo-passo fanno sempre parte della grande famiglia dei motori in corrente continua, però, a differenza di quelli a spazzole, i loro avvolgimenti sono esterni (statore) mentre i magneti sono interni (rotore). Quindi non necessitano di spazzole per portare corrente al rotore. Questo implica alcuni vantaggi tra cui l'assenza di usura, minimi disturbi (proprio per la mancanza dei contatti sulle spazzole), possibilità di bloccare il rotore in una specifica posizione; per contro, non è sufficiente collegare il motore ad un alimentatore ma bisogna comandarlo con un apposito driver.
Il driver deve essere in grado di fornire corrente alternativamente con una sequenza prestabilita su ogni fase del motore (di solito quattro) alla quale corrisponde la rotazione dell'albero motore di uno step (da cui, appunto, il nome di "step-motor"). A seconda del tipo di motore, si possono avere 64, 100, 200, 360 o più step con il vantaggio che è possibile mantenere il motore fermo in un determinato step semplicemente fermando la sequenza di alimentazione ma continuando a mantenere alimentato il motore (con una corrente impostabile sul driver di comando). Viste le esigue potenze realizzabili in pratica, questi motori vengono usati per movimenti di precisione in stampanti o bracci robotizzati per lo spostamento di piccoli oggetti. La gestione da parte di un microcontrollore risulta assai agevole, dovendo dialogare con un driver "intelligente" al quale è sufficiente indicare la direzione di rotazione e fornire un impulso per ogni step che si vuole ottenere. Per come funzionano, i motori passo-passo possono ruotare il proprio perno anche solo per porzioni di giro, con estrema precisione. I motori passo-passo si dividono in due categorie (unipolari e bipolari) a seconda della configurazione degli avvolgimenti interni. In quelli unipolari sono presenti quattro avvolgimenti e la corrente li percorre in un solo senso; il cablaggio consiste in quattro fili, uno per ciascun avvolgimento, più un filo comune (5 fili totali) oppure quattro fili, uno per ciascun avvolgimento più due fili ciascuno il comune di due avvolgimenti (6 fili totali). In quelli bipolari sono presenti solo due avvolgimenti ma la corrente può andare in entrambi i sensi; il cablaggio consiste in due fili per ciascun avvolgimento, per un totale di 4 fili.

cordiamo che in un motore elettrico oltre alla corrente assorbita in funzionamento normale è importante considerare anche la corrente massima richiesta in fase di avvio (corrente di spunto) che può arrivare anche a 3 o 4 volte la corrente nominale. Se prevedete un utilizzo intenso dei motori con continue accelerazioni e frenate, è importante che anche questa corrente non superi il valore consigliato dei 2 A. Per tale ragione sceglierete dei motori con corrente nominale non superiore a 500 mA.
Per i primi esperimenti abbiamo utilizzato un piccolo motore da 6 V (corrente massima di 100 mA) ricavato da un'automobilina giocattolo dismessa, cablato come visibile nella **Fig. 3**. Per l'alimentazione abbiamo usato un alimentatore universale non stabilizzato da 5 W, impostato per fornire una tensione di 6 V, sufficienti per alimentare sia la scheda ArduMoto che Arduino.
Come primo sketch facciamo in modo da attivare entrambi i motori con la seguente sequenza che si ripete all'infinito: motore A e B avanti a mezza velocità, motore A e B avanti a piena velocità, motore A e B fermi, motore A e B indietro a mezza velocità.
Una possibile ed interessante applicazione è il controllo delle ruote di una piattaforma robotica come quella commercializzata dalla ditta Futura Elettronica (codice 7300-2WDP-MA) perfettamente compatibile con le schede Arduino. L'assemblaggio di questa piattaforma robotica è davvero agevole, in quanto essa è già fornita di tutte le viti necessarie al fissaggio.
Per i più tecnici forniamo anche le caratteristiche elettriche dei motori, che trovate nella **Tabella 3**.
Il parametro "a rotore bloccato" si riferisce al valore della corrente assorbita dal motore quando la ruota è ferma, ovvero quando al robot da fermo viene data tensione per farlo partire o quando lo stesso robot dovesse andare a sbattere contro una parete rimanendovi bloccato. È in pratica la massima corrente assorbita dai motori ed anche il punto in cui i motori sono maggiormente sollecitati, ma come potete vedere, anche alla massima tensione di 6 volt, siamo ampiamente al disotto del valore dei 2 A sopportato dal driver di ArduMotor. Utilizzando la piattaforma robotizzata a quattro ruote, i motori di ogni lato

potranno essere connessi in parallelo funzionando all'unisono come in un carro armato, rispettando ancora una volta il limite massimo di 2A.
Per l'alimentazione ci affidiamo a 4 batterie ricaricabili da 1,2 V, le quali, completamente cariche, forniranno 1,5x4=6 volt (il massimo consentito dai motori). Tuttavia non potremo sfruttarle al massimo, perché già ad 1,1 volt per cella la tensione totale sarà di soli 4,4 V.
Inserite le batterie in un portabatterie e realizzate un cavetto di alimentazione che abbia da una parte il plug per la scheda Arduino e dall'altra la clip per il portabatterie. Sulla confezione della base robotica troverete anche tutte le viti per il fissaggio, un plug maschio ed un interruttore, utili per realizzare un cablaggio più raffinato.
Abbiamo messo appunto uno sketch apposito denominato *motor_4*, che permette di comandare i motori in direzione e velocità tramite comandi dal PC, ovviamente con il cavo di programmazione connesso.
Anche se in modo limitato, è comunque possibile testare le funzionalità della piattaforma, se non vi soddisfa il senso di rotazione è comunque possibile invertire i fili del motore interessato.
Ci rimane, adesso, un'ultima parte riguardante i motori passo-passo, non essendo disponibile una vera e propria motor shield specifica per questi motori abbiamo optato per l'utilizzo di un driver siglato *EasyDriver*, fornitoci dalla ditta Futura (codice 7300-EASYDRIVER) e reperibile sul sito *www.futurashop.it*.
Questo driver è basato sul chip A3967SLB della Allegro ed è in grado di controllare un singolo motore passo-passo bipolare con possibilità di selezionare quattro modalità di controllo del motore: passo, 1/2 passo, 1/4 di passo e 1/8 di passo. Consente di impostare la corrente in uscita tramite l'apposito trimmer (montato sul circuito).
La versione di EasyDriver usata in questo esempio è la 4.3 ed i rispettivi collegamenti sono illustrati nella **Tabella 4**.
Non dovete spaventarvi di tutti i contatti di cui dispone, in quanto il driver è già impostato in modo ottimale per la maggior parte delle applicazioni; l'unica raccomandazione è non collegare e scollegare il motore quando il driver è alimentato, per non danneggiare l'integrato A3967.
Il motore utilizzato in questo esempio è distribuito dalla ditta Futura Elettronica (codice 7300-STEPMOT01); esso ha le caratteristiche descritte nella **Tabella 5**.
In ogni caso, assicuratevi che il motore che vi accingete ad usare non assorba una corrente superiore a 750 mA e possa essere alimentato con una tensione compresa tra 8 e 30 volt, oltre ad essere di tipo bipolare.
Per sapere se il motore sia effettivamente bipolare è sufficiente verificare se ha almeno quattro fili; allo scopo utilizzate un tester e misurate la continuità tra i vari fili per identificare i due avvolgimenti A e B. Se avete recuperato il motore in qualche discarica e non

Angolo passo-passo	1,8° (200 passi)
Numero di fasi	2 (bipolare)
Resistenza per fase	55 ohm
Induttanza per fase	80 mH
Resistenza d'isolamento	100 Mohm min. (500 Vcc)
Classe d'isolamento	B
Inerzia del rotore	54 g.cm²
Massa	0,23 kg
Alimentazione	max. 15,4 V
Consumo	0,28 A
Coppia di tenuta (coppia che, con motore alimentato, si oppone alla rotazione)	2,4 kg x cm
Coppia residua (coppia che si oppone alla rotazione dell'albero di un motore non alimentato)	120 g x cm
Dimensioni	42,3 x 42,3 x 37 mm

***Tabella 5** – Caratteristiche del motore 7300-STEPMOT01 della Futura Elettronica.*

Pin	Collegamento
Motor A	Avvolgimento A
Motor A	Avvolgimento A
Motor B	Avvolgimento B
Motor B	Avvolgimento B
GND	Massa alimentazione motori. Da collegarsi al pin GND di Arduino.
M+	Alimentazione positiva motori. Da collegarsi al pin vin di Arduino.
Cur Adj	Regoliamola al minimo, risparmieremo corrente, anche se il motore avrà meno forza a rotore fermo.
GND	Massa segnale di comando. Da collegare al pin GND della scheda Arduino.
STEP	Impulso di comando corrispondente ad un step. Da collegare al pin 9 della scheda Arduino.
DIR	Imposta la direzione di rotazione. Da collegare al pin 10 della scheda Arduino.

***Tabella 6** – Connessioni di EasyDriver da usare per la nostra applicazione. Tutti gli altri pin non sono utilizzati.*

Listato 3

```
int StepPin = 9;
int DirPin  = 10;

void setup() {
  pinMode(StepPin, OUTPUT);
  pinMode(DirPin, OUTPUT);
}

void loop(){
  delay(1000);
  Rotate(false,1000, 1250);   // Richiama funzione per la rotazione
}

void Rotate(boolean dir,int steps, int Delay){
  // Dir direzione true o false
  // Steps numero di impulsi
  // Delay ritardo tra un impulso ed il successivo

  digitalWrite(DirPin,dir);
  delay(50);
  for(int i=0;i<steps;i++){                        // ripete steps volte
        digitalWrite(StepPin, HIGH);         // pone alto il pin
        delayMicroseconds(Delay/2);         // attende
        digitalWrite(StepPin, LOW);          // pone basso il pin
        delayMicroseconds(Delay/2);         // attende
  }
}
```

trovate i suoi dati, assicuratevi almeno che la resistenza di ogni fase sia sufficientemente alta per poter garantire una corrente non superiore ai 750 mA. Ipotizzando una tensione di alimentazione di 12 volt, la resistenza di ciascuna fase dovrà essere di almeno 16 ohm; se trovate un valore più basso, il motore non è adatto al nostro driver.
Appurata l'idoneità del motore, non vi resta che cablare il tutto seguendo le indicazioni riportate nella **Tabella 6**; in essa, per comodità ai pin del driver EasyDriver sono stati saldati alcuni strip maschio per poterlo utilizzare con una piccola breadboard. Facciamo in modo che l'alimentazione tramite il plug di Arduino possa alimentare anche EasyDriver, sfruttando il pin Vin allo scopo. Per far funzionare l'insieme, è sufficiente utilizzare il solito alimentatore non stabilizzato impostato per una tensione di uscita di 6÷9 V. Un pezzetto di nastro adesivo renderà chiaramente visibile la rotazione del perno del motore.

Cablaggio del motore passo-passo con ArduMoto ed EasyDriver.

Per quanto riguarda il programma di test, è sufficiente che generi una sequenza di impulsi alla frequenza desiderata, per il numero impostato. Lo sketch prevede una procedura denominata *Rotate*, alla quale passare come parametri la direzione, il numero di impulsi ed il ritardo tra un impulso ed il successivo. Il suo utilizzo è molto semplice: se, ad esempio, con il motore in nostro possesso volessimo compiere un giro intero, non dovremmo fare altro che generare 1.600 impulsi. Questo perché il motore è un 200 step, ognuno dei quali è ampio 1,8 °; sapendo che tramite i pin MS1 e MS2, Easydriver è impostato ad 1/8 di passo, vediamo che effettivamente per un giro (formato da 200 passi) servono 200·8=1600 impulsi. Volendo compiere un giro in due secondi, otteniamo anche la variabile Delay: Delay=2/1.600=1.250 µs.
In generale Delay=(durata della rotazione)/(numero di impulsi) e steps=(Numero di Giri)/1600.
Il **Listato 3** espone il codice necessario alla gestione del motore e impiegante la variabile Delay. La libreria "stepper.h" non è compatibile con questo driver, ma è invece adatta ad un driver più semplificato, senza logica, in quanto la sequenza dell'alimentazione delle fasi del motore viene generata internamente dal firmware; sul sito ufficiale Arduino potrete trovare alcuni esempi applicativi in cui è specificato il tipo di hardware da usarsi. ■

Conoscere e usare Arduino

dell'ing. MIRCO SEGATELLO

Scopriamo Processing, il linguaggio di programmazione Java-based e open-source, col quale realizzeremo diverse applicazioni. Quinta puntata.

Processing è un linguaggio di programmazione basato su Java, che consente di sviluppare diverse applicazioni come giochi, animazioni e contenuti interattivi. Basandosi su Java, ne eredita completamente la sintassi, i comandi e il paradigma di programmazione orientata agli oggetti; in più, mette a disposizione numerose funzioni ad alto livello per gestire facilmente l'aspetto grafico e multimediale. É distribuito sotto licenza Open Source ed è supportato dai sistemi operativi GNU/Linux, Mac OS X e Windows.
Il pacchetto, scaricabile gratuitamente dal sito ufficiale *http://processing.org*, mette a disposizione un ambiente di sviluppo integrato (IDE), e le varie creazioni (chiamate sketch) vengono organizzate in uno sketchbook. Ogni sketch contiene in genere, oltre alle classi che lo compongono, una cartella chiamata *Data* in cui viene inserito il materiale multimediale utile all'applicazione, quali, ad esempio, immagini, font e file audio. Ogni applicazione creata può inoltre essere esportata come Java applet.
Le funzionalità di processing possono essere espanse tramite delle librerie aggiuntive, spesso create da terze parti, ottenendo un ambiente di sviluppo molto poliedrico. Ad esempio, è possibile utilizzare in Processing protocolli di comunicazione Seriale, TCP/IP, UDP, RDP, OSC, riuscendo in questo modo a gestire una moltitudine di applicazione hardware esterne. Grazie alla libreria specifica Arduino, l'in-

Fig. 1

```
arduino_output | Processing 1.2.1
File Edit Sketch Tools Help
arduino_output

import processing.serial.*;

import cc.arduino.*;

Arduino arduino;

color off = color(4, 79, 111);
color on = color(84, 145, 158);

int[] values = { Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LO
 Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW,
 Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW, Arduino.LOW };

void setup() {
  size(470, 200);

  println(Arduino.list());
  arduino = new Arduino(this, Arduino.list()[0], 57600);

  for (int i = 0; i <= 13; i++)
```

Fig. 2

```
NullPointerException

Stable Library
=========================================
Native lib Version = RXTX-2.1-7
Java lib Version   = RXTX-2.1-7
[0] "COM1"
[1] "COM2"
[2] "COM20"
[3] "COM21"
[4] "COM22"
[5] "COM23"
[6] "COM24"
[7] "COM25"
[8] "COM26"
[9] "COM27"
[10] "COM28"
[11] "COM29"
```

Fig. 3

tegrazione tra il mondo software e quello hardware non è mai stata così facile: basti pensare che l'ambiente di sviluppo Software di Arduino è derivato proprio da Processing, con il quale condivide sia la struttura che il linguaggio di programmazione. In questo articolo non ci addentreremo nella descrizione di Processing, peraltro ben documentato su diversi siti, ma piuttosto impariamo quei pochi ed essenziali passaggi che ci permettono di poterlo interfacciare con la scheda Arduino (la Duemilanove, in questo caso...) tramite alcuni semplici esempi. Lasciamo a voi il piacere di utilizzare questi strumenti come meglio vi conviene, realizzando degli applicativi più complessi e di maggiore utilità.

Per prima cosa dobbiamo preparare la nostra scheda Arduino in modo che possa essere gestita in remoto dal software Processing; allo scopo, dobbiamo collegarla al PC e caricarvi l'apposito sketch denominato *StandardFirmata.pde*, situato nella cartella *arduino-018\libraries\firmata\example\StandardFirmata.pde.*

Con il termine *Firmata* si intende un generico protocollo di comunicazione per microcontrollori e computer, che essendo molto semplice, può essere facilmente implementato su qualsiasi piattaforma software.

A chi volesse approfondire la conoscenza di questo protocollo, consigliamo il link: *http://firmata.org/wiki/Main_Page*. Il riferimento ad Arduino, invece, lo si trova all'indirizzo web *http://arduino.cc/en/Reference/Firmata*, dal quale è possibile scaricare, oltre alla libreria, anche alcuni esempi.

Caricando questo firmware, si ha la possibilità di gestire in remoto la scheda Arduino proprio con questo protocollo; tra le altre cose, il firmware è facilmente adattabile alle più svariate applicazioni; infatti nella cartella esempi troverete altri sketch che permettono di gestire funzioni specifiche della scheda Arduino.

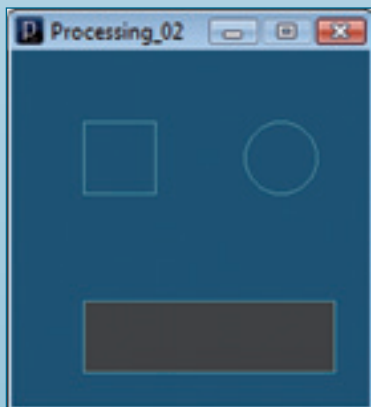
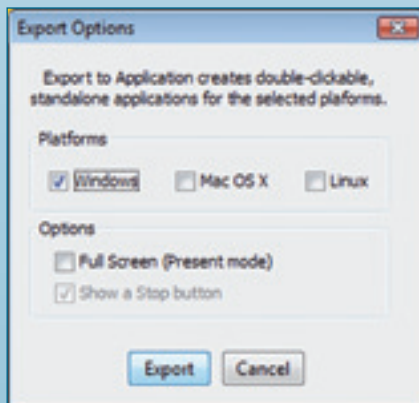
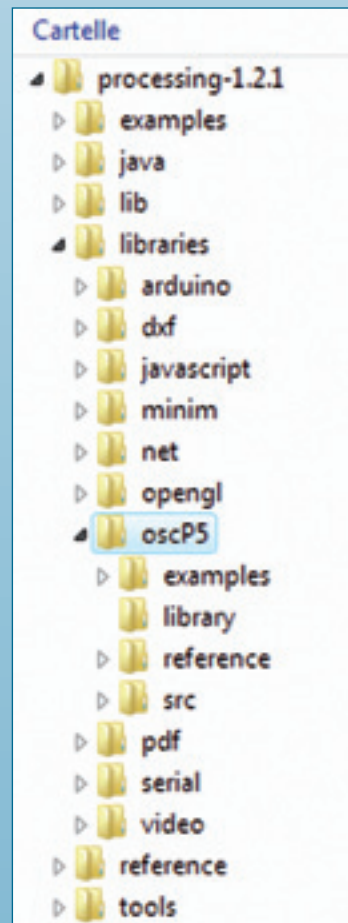
Prima di caricare lo sketch su Arduino assicuratevi che, nella riga **Firmata.begin(57600)**;, sia impostata una velocità di comunicazione di 57.600 bps. Per collaudare i nostri esempi collegate al pin 9, tramite una resistenza da 200 ohm, un LED, mentre al pin 2 (che useremo come ingresso) connettete un pulsante con la relativa resistenza di pull-down (**Fig. 1**).

Listato 1

```
/*
  processing_02

  Semplice esempio di utilizzo del software Processing per
  dialogare con Arduino duemilanove. Su Arduino è caricato
  StandardFirmata.pde, questo programma va utilizzato con Processing-1.2.1
  Con il mouse sul shape-quadrato si attiva l'uscita 13
  L'ingresso 2 viene usato per cambiare colore al shape-cerchio.
  Con il mouse sullo shape rettangolo si regola la luminosità
  del LED connesso all'uscita 9.

*/

import processing.serial.*;
import cc.arduino.*;
Arduino arduino;

color off = color(4, 79, 111);
color on = color(84, 145, 158);

int[] values = { Arduino.LOW };
int pin_out = 13;  // Definisce pin di uscita 13 (LED on board)
int pin_in  = 2;   // Definisce pin di ingresso 2

void setup() {
  size(200, 200);

  println(Arduino.list());
  arduino = new Arduino(this, Arduino.list()[2], 57600);

  // Impostare l'indice array (2) a seconda della COM usata da Arduino
  // Utilizzare le indicazioni riportate sulla taskbarr in basso all'avvio
  // del programma

  arduino.pinMode(pin_out, Arduino.OUTPUT);
  arduino.pinMode(pin_in, Arduino.INPUT);

}

void draw() {

  background(off);
  stroke(on);

    // Modifica colore shape-qadrato a seconda del livello logico dell'uscita
    if (values[0] == Arduino.HIGH)
      fill(on);
    else
      fill(off);
    rect(40, 40, 40, 40);

    // Modifica colore shape-cerchio a seconda del livello logico dell'ingresso
    if (arduino.digitalRead(pin_in) == Arduino.HIGH)
      fill(on);
    else
      fill(off);
    ellipse(150, 60, 40, 40);

    // Controllo luminosità LED
    fill(constrain(mouseX / 2, 0, 255));
    rect(40, 140, 140, 40);
    arduino.analogWrite(9, constrain(mouseX / 2, 0, 255));
  }


void mousePressed()
{
  // Verifica sel premuto pulsante del mouse sopra la shape-quadrato
  // nel qual caso inverte stato pin di uscita

  if (mouseX>40 & mouseX<80 & mouseY>40 & mouseY<80)  {

    if (values[0] == Arduino.LOW) {
      arduino.digitalWrite(pin_out, Arduino.HIGH);
      values[0] = Arduino.HIGH;
    } else {
      arduino.digitalWrite(pin_out, Arduino.LOW);
      values[0] = Arduino.LOW;
    }
  }
}
```

Fig. 4

Fig. 5

Fig. 6

A questo punto passiamo al lato PC; per prima cosa dobbiamo scaricare e scompattare Processing dal sito di riferimento: *http://processing.org/download/*. Appena scompattato, il software è subito pronto e non necessita di installazione, proprio come l'IDE di Arduino. Affinché Processing possa dialogare con Arduino, è necessario installare l'apposita libreria che troverete qui: *http://www.arduino.cc/playground/uploads/Interfacing/processing-arduino-0017.zip*. Esiste anche una sezione del sito di Arduino dedicata a Processing, rintracciabile dalla pagina web *http://www.arduino.cc/playground/Interfacing/Processing*.
Esistono diverse versioni di questa libreria; ad esempio, una meno recente si trova in un file chiamato *processing-arduino2.zip*, ma consigliamo di utilizzare sempre la libreria più recente. Scompattate i file scaricati e copiate la cartella *Arduino* all'interno della cartella *libraries* di Processing; l'aggancio alla libreria avviene al primo avvio del programma.
All'interno dei file della libreria trovate anche alcuni esempi di sketch per Processing, con alcuni controlli per Arduino; ad esempio, *arduino_output.pde* (**Fig. 2**) disponibile in *processing/libraries/arduino/example*, che permette di comandare le uscite di Arduino.
Con la scheda Arduino connessa alla USB, caricate questo sketch su Processing e verificate la riga *arduino = new Arduino(this, Arduino.list()[0],* ***57600****)*. Assicuratevi che la velocità di comunicazione sia impostata su *57600*; ricordate che occorre impostare la porta di comunicazione settando l'indice del vettore *Arduino.list*, il quale contiene l'elenco delle porte COM installate nel PC. Andate su esplora risorse di Windows per conoscere su quale porta è presente Arduino, quindi avviate lo sketch; anche se ciò determinerà la segnalazione di errore in comunicazione, vi farà vedere nella taskbar in basso le porte COM installate ed il corrispondente indice del vettore.
Avviate il programma, la prima casella a sinistra corrisponde all'uscita 13 di Arduino, alla quale è presente il LED interno. Potete accenderlo e spegnerlo a vostro piacimento. Per verificare le altre uscite è necessario connettere un LED su ogni uscita.
Sono presenti altri due esempi utili per testare uno gli ingressi digitali e l'altro le uscite PWM.
Per provare appieno queste nuove funzioni, ci siamo cimentati nella stesura di uno sketch che permettesse di attivare un'uscita e visualizzare lo stato di un ingresso; lo abbiamo chiamato *Processing_01.pde*. Descriviamo, però, un secondo sketch denominato *Processing_02*, con il quale oltre a gestire in ed out controlliamo la luminosità di un LED tramite l'uscita PWM. Le righe di codice corrispondenti sono riepilogate nel **Listato 1**.
La **Fig. 4** mostra la schermata di uno sketch che comanda i LED della scheda Arduino: facendo clic nella casella quadrata in alto a sinistra, attivate il LED interno ad Arduino, mentre sul cerchio in alto a destra potete vedere il livello logico del pin 2, usato come ingresso. Il rettangolo in basso, se puntato col mouse permette di regolare la luminosità del

LED connesso all'uscita 9.
Con un po' di pratica, seguendo i numerosissimi tutorial ed esempi disponibili in rete, potrete realizzare la vostra interfaccia; vi ricordiamo che il software è in grado di gestire anche grafica in 3D ed animazioni. L'applicativo può essere esportato in svariati formati, anche come applet java da inserire nel vostro sito personale. È anche possibile esportare l'applicazione in formato eseguibile (**.exe**) per i diversi sistemi operativi, in modo da distribuire molto facilmente il vostro lavoro.

OSC

Se già quanto esposto finora vi sembra abbastanza, adesso vi faremo vedere un'altra interessantissima applicazione realizzata con questo software. Si tratta di utilizzare il protocollo OSC per poter gestire la scheda Arduino tramite una periferica esterna con accesso alla rete in modalità Wi-Fi.
Ma andiamo per gradi e spendiamo due parole per spiegare cos'è OSC. Open Sound Control (OSC) è un protocollo di comunicazione per messaggi tra computer, strumenti musicali elettronici e altri dispositivi multimediali, ottimizzato per funzionare nelle moderne reti informatiche. I benefici delle moderne tecnologie di reti nel mondo degli strumenti elettronici OSC permette di aggiungere molta flessibilità organizzativa oltre, ovviamente, a controlli centralizzati per grossi studi, teatri o concerti. Per chi volesse conoscere più approfonditamente questo standard di comunicazione consigliamo il sito: *http://opensoundcontrol.org/*.
Affinché Processing possa gestire una comunicazione con OSC è necessario installare l'apposita libreria, scaricabile all'indirizzo *http://www.sojamo.de/libraries/oscP5/*. Sempre a questo indirizzo, è disponibile tutta la documentazione per l'utilizzo di tale libreria. Il file *oscP5.zip* deve essere scompattato all'interno della cartella *libraries* di Processing (**Fig. 6**).
Al primo avvio di Processing la libreria verrà caricata e sarà subito utilizzabile.
Quello che vogliamo fare è utilizzare questa libreria per dialogare con un iPhone connesso in Wi-Fi alla rete del PC e poter così gestire da remoto la scheda Arduino. Per arrivare a fare questo è necessario installare sull'iPhone un'applicazione in grado di utilizzare lo standard Open Sound Control che si chiama TouchOSC, il cui sito di riferimento è *http://hexler.net/touchosc*.

Fig. 7

Fig. 8

Fig. 9

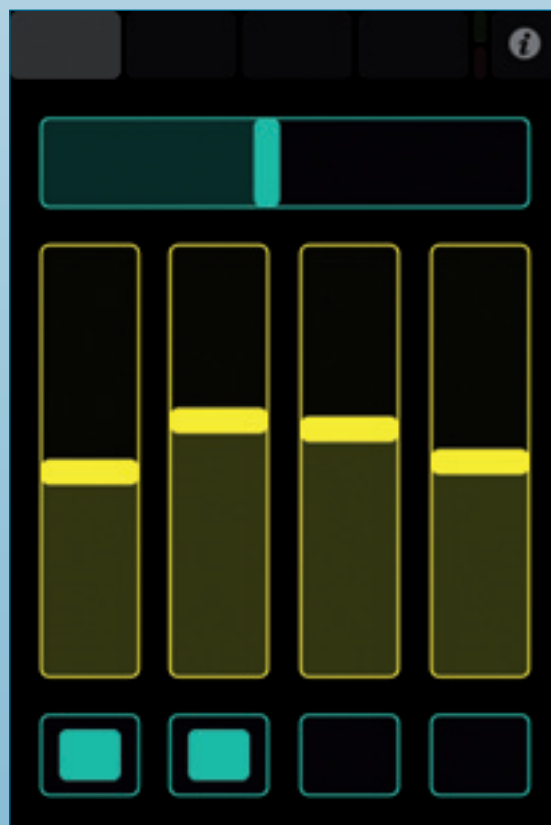



Fig. 10

Dal sito è possibile scaricare il manuale utente *touchosc-manual-v1-1.pdf* ed alcuni esempi di utilizzo anche per Processing:

- *simple.pde*; serve alla ricezione ed alla visualizzazione di messaggi dal "Simple" layout (solo pagina 1); richiede la libreria *oscP5*;
- *Example Code* per la gestione dei dati dell'accelerometro;
- *Processing Apps* – *Processing examples* + video by Mike Cook.

TouchOSC è scaricabile dall'AppStore della Apple, al costo di 3,99 euro, giustificato anche dal fatto che questa applicazione è completamente configurabile tramite un semplicissimo editor, che può essere scaricato dal sito di riferimento e che è:

- touchosc-editor-1.4-osx.zip per Mac OS X;
- touchosc-editor-1.4-win32.zip per Windows;
- touchosc-editor-1.4-noarch.zip per Linux o altri sistemi operativi.

Fig. 11a

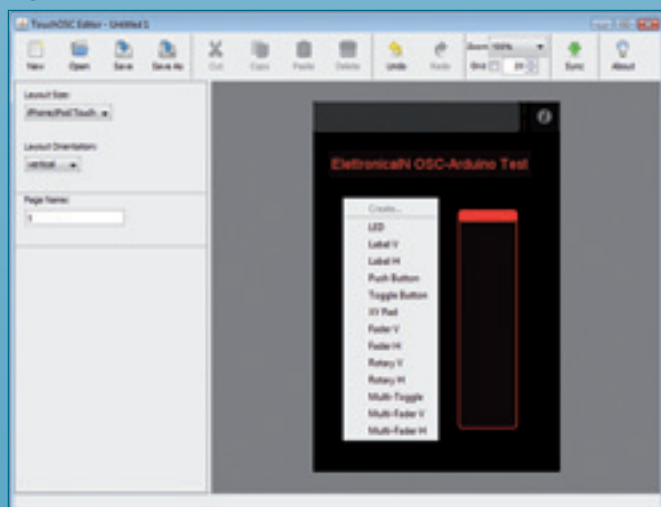
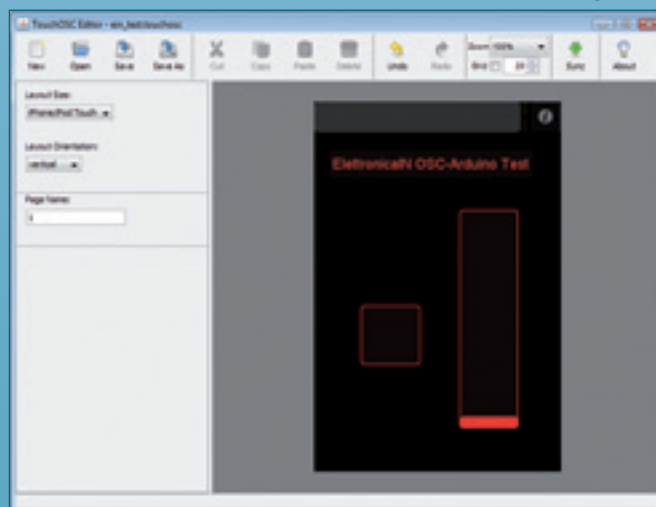



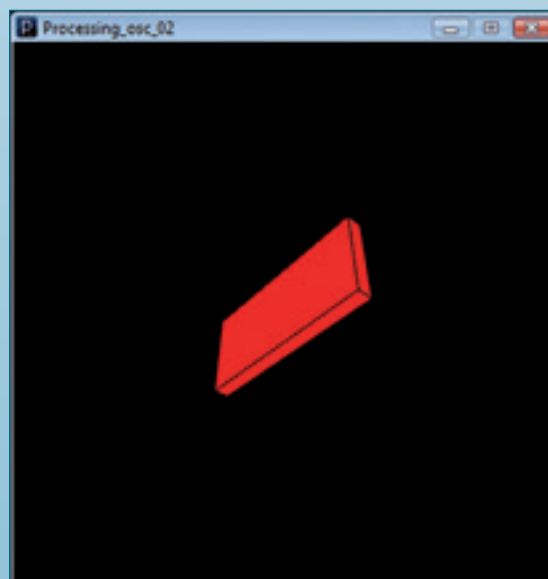
Fig. 11b

Per mettere in funzione il tutto è bene iniziare con l'esempio,denominato *simple.pde*, già disponibile sul sito. Per poterlo utilizzare dovete creare una cartella chiamata *simple* all'interno della libreria oscP5 e copiarci dentro il file *simple.pde*.
Create quindi una rete Wi-Fi tra il PC e l'iPhone (**Fig. 7**).
Aprite processing ed avviate lo sketch *simple.pde*; dal lato iPhone avviate l'applicativo TouchOSC ed impostate i parametri per l'accesso alla rete mediante il protocollo UDP (figure 8 e 9).
Fatto ciò, alla voce HOST inserite l'indirizzo IP del Computer, quindi nella sezione PORT (outgoing) scrivete il numero 8000; alla voce PORT (incoming) immettete il numero 9000. Introdotti questi valori, alla voce LOCAL IP ADRESS comparirà in automatico l'indirizzo IP dell'iPhone.
Nella relativa schermata selezionate come Layout il modello *Simple* ed avviate l'applicativo (**Fig. 10**).
Complimenti! Adesso il vostro iPhone potrà interagire con il software Processing.
Adesso siamo pronti per creare il nostro applicativo per la gestione della scheda Arduino: vogliamo realizzare un controllo con un pulsante per accendere e spegnere un LED ed un fader per regolare la luminosità di quest'ultimo.
Mentre su Arduino è sempre caricato *firmataStandard.pde*, noi dobbiamo occuparci di creare l'interfaccia per l'iPhone e lo sketch per Processing.
Avviamo quindi *TouchOSC editor* e, sulla schermata dell'iPhone, cliccando con il

Listato 2

```
/**
  processing_osc

  Test controllo Arduino con IPhone
  su Arduino è installato firmatastandard
  V1.0

 */

// Per OSC
import oscP5.*;
import netP5.*;
OscP5 oscP5;
// Per Arduino
import processing.serial.*;
import cc.arduino.*;
Arduino arduino;

int[] values = { Arduino.LOW };
int pin_out = 13;  // Definisce pin di uscita 13 (LED on board)

float fader1 = 0.0f;
float toggle1 = 0.0f;
float led1 = 0.0f;

void setup() {
  size(320,440);
  frameRate(25);
  /* start oscP5, listening for incoming messages at port 8000 */
  oscP5 = new OscP5(this,8000);
  // Impostazioni per Arduino
    println(Arduino.list());
    arduino = new Arduino(this, Arduino.list()[3], 57600);

  // Impostare l'indice array (2) a seconda della COM usata da Arduino
  // Utilizzare le indicazioni riportate sulla taskbarr in basso all'avvio
  // del programma

    arduino.pinMode(pin_out, Arduino.OUTPUT);
}

void oscEvent(OscMessage theOscMessage) {

    String addr = theOscMessage.addrPattern();
    float  val  = theOscMessage.get(0).floatValue();

    if(addr.equals("/1/fader1"))        { fader1 = val; }
    else if(addr.equals("/1/toggle1"))  { toggle1 = val; }
}

void draw() {
    background(0);

    // toggle 1 outlines
    fill(0);
    stroke(0, 196, 168);  // Set colore bordo pulsante
    rect(80,95+100,60,60);  //Set posizione bordo pulsante

    // toggle 1 fills
    fill(0, 196, 168);
    if(toggle1 == 1.0f) {
      rect(80,95+100,60,60);
      arduino.digitalWrite(pin_out, Arduino.HIGH);  // LED ON
      }
    else
      arduino.digitalWrite(pin_out, Arduino.LOW); // LED OFF

    // fader 1 outlines
    fill(0);
    stroke(255, 237, 0);    // Set colore bordo fader
    rect(20+200,95,60,255);  //Set posizione bordo fader

    // fader 1 fills
    fill(255, 237, 0);
    rect(20+200,95+200+55,60,-fader1*255);  //Set posizione bordo fader
    arduino.analogWrite(9, int(fader1*255));  //Set luminositÃ  LED

}
```

Fig. 12

Fig. 13

pulsante destro inseriamo un toggle button ed un fader verticale (**Fig. 11a-b**). Cliccate su *sync* e seguite le istruzioni a video per caricare la nuova schermata sull'iPhone; l'operazione avviene tramite la rete Wi-Fi in modo molto semplice e funzionale. Selezionate questo nuovo layout ed avviatelo, aprite processing e caricate lo sketch *processing_osc_01.pde* che riportiamo nel **Listato 2**. Lo sketch in questione prevede la gestione dei messaggi in standard OSC provenienti dall'iPhone e intercettati grazie alla funzione *oscEvent*, dai quali viene estrapolato il valore del fader e del pulsante. Questi valori sono utilizzati per gestire la grafica a video (anche se non strettamente necessario) e i comandi per Arduino, già sperimentati nei precedenti esempi. Ricordatevi di attivare la casella stay connected per fare in modo che il canale di comunicazione rimanga aperto (**Fig. 12**).
Un ultimo esempio molto significativo (*processing_osc_03.pde*) prevede l'utilizzo dell'accelerometro interno all'iPhone i cui valori possono essere elaborati tramite Processing, in questo caso per comandare la luminosità del LED della scheda Arduino. Per questa applicazione è importante che attiviate la casella Accelerometer, questo permetterà all'applicazione TouchOSC di inviare in continuazione i valori dell' accelerometro interno ad IPhone. Nella schermata dello sketch di processing sarà visualizzato un rettangolo in 3D che mostrerà l'orientamento dell'IPhone, mentre, a seconda della sua inclinazione, si potrà variate la luminosità del LED connesso al pin 9 della scheda Arduino. Il LED dell'uscita 13, invece, si accenderà e si spegnerà a seconda di quale faccia dell'iPhone è rivolta verso l'alto (**Fig. 13**). Completiamo l'articolo con un esempio conclusivo che prevede l'utilizzo di comandi OSC bidirezionali tra Processing (file *processing_osc_04*) e ToucOSC (file *ein_testin.touchosc*). Oltre ai comandi già descritti, che prevedono da IPhone la gestione di un pulsante e uno slider di Processing, adesso aggiungiamo un controllo LED su Touchosc la cui accensione è gestita da Processing. Non riportiamo l'intero codice, ma solo la nuova parte attinente l'invio di comandi.
La prima riga dichiara una variabile IPhoneLocation che contiene l'indirizzo IP del nostro IPhone e la porta di comunicazione adibita alla ricezione dei messaggi, già dichiarata su Touchosc:

```
IPhoneLocation = new NetAddress("192.168.0.46",9000);
```

Quando viene premuto il pulsante del mouse su un qualsiasi punto dello sketch di Processing avviene l'invio del comando per attivare il LED:

```
void mousePressed() {
   OscMessage myMessage = new OscMessage("/1/led1");
```

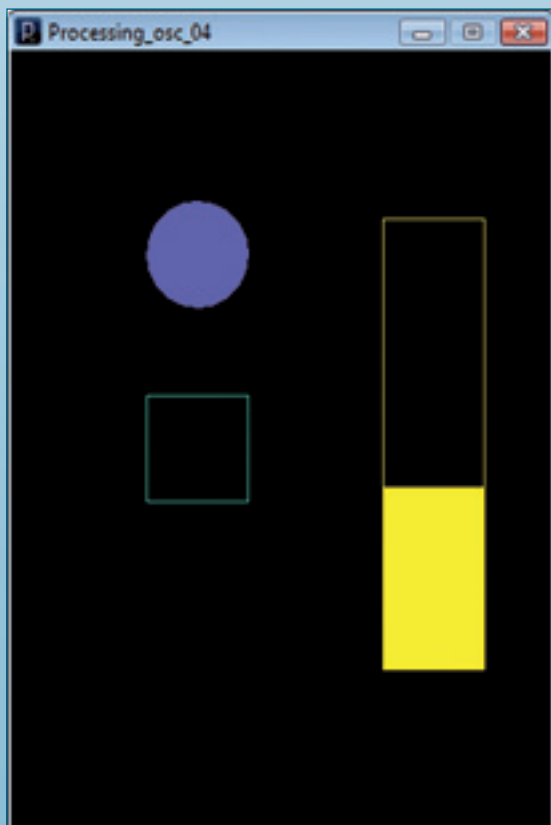



Fig. 14

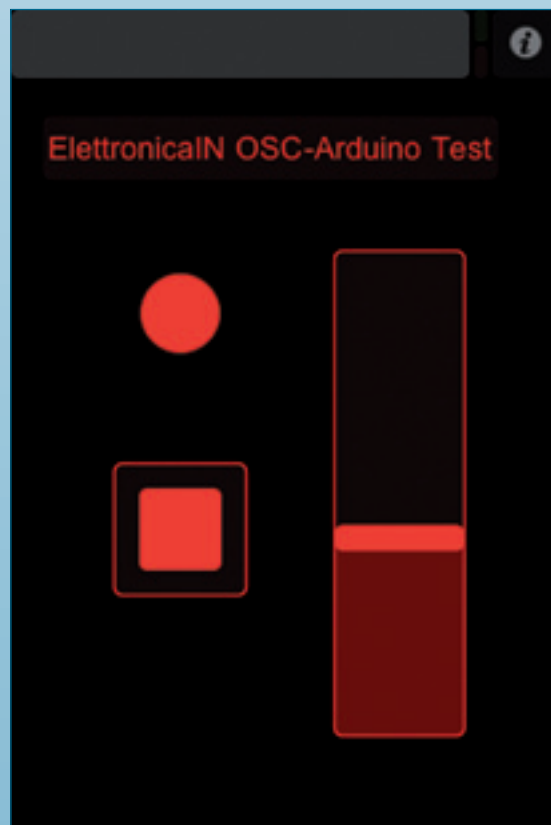



Fig. 15

```
  if (LEDStatus==1) {
    LEDStatus=0;
    myMessage.add(LEDStatus); /* add comando per
led1 */
    }
  else  {
    LEDStatus=1;
    myMessage.add(LEDStatus); /* add comando per
led1 */
  }

  oscP5.send(myMessage, IPhoneLocation);
}
```

La prima riga provvede alla creazione del messaggio che deve riportare la posizione del LED ("/1/led1"); successivamente al messaggio, viene aggiunto (*mymessage.add*) il valore, in questo caso 1 o 0, a seconda se si vuole accendere o spegnere il LED. Dopodiché il messaggio viene spedito all'indirizzo specificato. L'istruzione IF viene semplicemente usata per la funzione toggle del LED ad ogni pressione del pulsante del mouse (**Fig. 14-Fig. 15**).
L'aggiunta delle righe di codice per la gestione di Arduino risulta abbastanza semplice; prendete spunto dagli esempi precedenti.

Conoscere e usare Arduino

dell'ing. MIRCO SEGATELLO

Impariamo ad utilizzare i moduli XBee per realizzare semplici collegamenti Wireless nelle nostre applicazioni con Arduino. Sesta puntata.

In queste pagine vedremo come sia possibile dotare la scheda Arduino di un modulo di comunicazione radio, al fine di comunicare con un'altra scheda Arduino oppure con un PC. Tra le infinite possibilità offerte dal mercato, Arduino ha sposato lo standard ZigBee e quindi le applicazioni si sono orientate verso uno specifico hardware, comunque disponibile a livello commerciale in molteplici versioni.

LO STANDARD ZIGBEE

Il termine ZigBee deriva dalle parole inglesi "zigging bee", che significano "danza delle api"; è stato adottato ispirandosi al modo di comunicazione essenziale e veloce che tali insetti utilizzano per la sopravvivenza di ogni loro colonia. Il protocollo ZigBee è stato implementato dalla ZigBee Alliance (*www.zigbee.org/en/index.asp*) che è un consorzio no-profit per la fabbricazione di semiconduttori, obiettivo del quale è stato creare un protocollo molto versatile a larga diffusione in grado di funzionare in strutture di reti multiple, caratterizzate da un basso consumo e adatte ad essere alimentate a batteria per lunghi periodi. È il caso di telemetria, sistemi di allarme, domotica, rilevatori di fumo ecc. Il protocollo ZigBee è stato studiato per poter implementare diverse tipologie di reti, statiche, dinamiche, a stella e mesh, con la possibilità di arrivare sino a 65.000 nodi e garantendo l'assenza di collisioni ed un controllo degli errori molto avanzato.

Tabella 1 - *Confronto tra gli standard di comunicazione.*

Standard	ZigBee 802.15.4	Wi-Fi 802.11b	Bluetooth 802.15.1
Portata in metri	1 ÷ 100	1 ÷100	1 ÷ 10
Durata batteria	100 ÷ 1000	0,5 ÷ 5	1 ÷ 7
Numero di nodi nella rete	65.000	32	7
Applicazioni	Monitoraggio, controllo, telemetria	Web, Email, Video	In sostituzione di cablaggio
Stack Size (KB)	4 ÷ 32	1.000	250
Velocità di trasmissione (kb/s)	20 ÷ 250	11.000	720

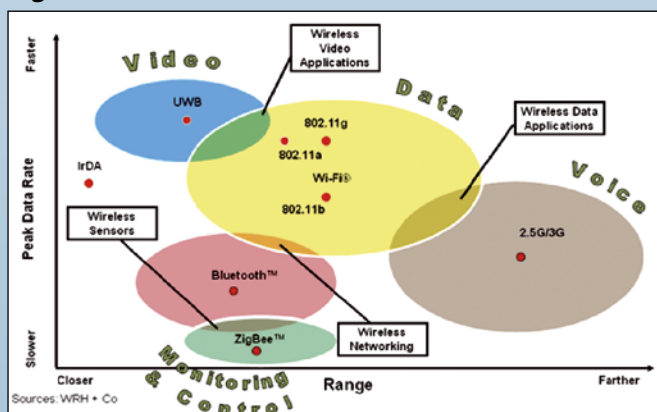
Fig 1 - *Confronto tra i vari standard di comunicazione.*

Il protocollo ZigBee prevede delle specifiche di sicurezza per l'accesso alla rete, tra cui un sistema criptato a 128 bit basato sul *NIST Certified Advanced Encryption Standard* (AES).
Confrontando lo ZigBee con il Wi-Fi ed il Bluetooth, ci si accorge subito di alcune differenze: il Bluetooth è essenzialmente utilizzato per far colloquiare due apparati a brevissima distanza, senza l'utilizzo di cavi; di solito è implementato sui telefoni cellulari per utilizzare periferiche esterne come auricolari e ricevitori GPS o per lo scambio dati con il PC. Non è in grado di gestire molte periferiche contemporaneamente ed il raggio di portata che consente è piuttosto limitato.
Il Wi-Fi ha il vantaggio di una portata maggiore, tanto che con apparati ed antenne specifiche si arriva sino a qualche chilometro; ha una velocità di trasferimento notevole, migliorata con gli ultimi standard, che gli permette l'invio di immagini in streaming e consente la creazione di reti anche complesse sia tra periferiche, sia fra PC. Tuttavia gli alti costi dell'hardware e l'elevato consumo di corrente non lo rendono idoneo in applicazioni a basso costo con alimentazione a batteria per connessioni di lunga durata (**Tabella 1**).
A livello commerciale, lo standard ZigBee/IEEE 802.15.4 è stato implementato dalla Maxstream-Digi (*www.maxstream.net e www.digi.com*) con i moduli denominati XBee.
I primi di questi moduli immessi nel mercato e denominati semplicemente XBee, si riferiscono alla serie 1 ed implementano solo in parte il protocollo ZigBee, mantenendo, però, la caratteristica di poter essere utilizzati nella realizzazione di reti a basso costo e a basso consumo. I moduli sono semplici da utilizzare, richiedono pochissima energia e costituiscono una soluzione efficace ed affidabile per la trasmissione di dati critici. I moduli XBee operano nella banda ISM alla frequenza di 2,4 GHz, permettendo di realizzare applicazioni hardware estremamente compatte; inoltre sono compatibili con le normative vigenti in paesi come U.S.A., Canada, Australia, Israele e negli stati dell'Europa. Ciascun modulo è formato da un transceiver a radiofrequenza ed un microcontrollore, con firmware aggiornabile, che gestisce, oltre alla comunicazione radio, anche una serie di linee di I/O sia digitali che analogiche, con funzioni di interrupt e sleep. Ciò permette di connettere direttamente al modulo sensori o contatti e di farlo lavorare sul campo in modalità stand-alone, senza la necessità di interfacciarlo con altri elementi esterni se non una batteria di alimentazione. Tutte le funzioni che dovrà svolgere saranno programmate dall'utente secondo le esigenze. Di questa prima serie esiste anche la versione PRO, che pur mantenendo le medesime caratteristiche e la stessa piedinatura (cambia la lunghezza) ha una potenza trasmissiva maggiore, che permette la comunicazione a

Tabella 2 - *Caratteristiche tecniche dei moduli Xbee serie 1.*

Parametro	XBee	XBee-PRO
Frequenza operativa	ISM 2,4GHz	ISM 2,4GHz
Potenza in trasmissione	1 mW	63 mW
Sensibilità ricezione	-92 dB	-100 dB
Portata	30 m (indoor) 100 m (in aria libera)	90 m (indoor) 1.600 m (in aria libera)
Consumo	TX 45 mA RX 50 mA Power-down < 10µA	TX 250 mA RX 55 mA Power-down < 10µA
Velocità di trasferimento RF	250 kbps	250 kbps
Velocità trasferimento UART	1,2÷115 kBaud	1,2÷115 kBaud
Estensioni	Linee di A/D e digitali	Linee di A/D e digitali
Periferiche indirizzabili	65.000	65.000
Numero di canali disponibili	16	12
Alimentazione	2,8÷3,4V	2,8÷3,4V

grande distanza; in sintesi, le caratteristiche tecniche sono riportate nella **Tabella 2**.
Esistono diverse versioni di moduli XBee a seconda del tipo di antenna adottato, il più performante dei quali consente l'utilizzo di un'antenna esterna con attacco U.FL; ma i costi e gli ingombri maggiorati non sempre ne giustificano l'utilizzo. Una seconda versione prevede un'antenna a stilo già installata e consente di ottenere quasi la massima portata dichiarata; un'ultima versione dispone di un'antenna integrata, decisamente molto pratica, ma che riduce la portata di circa il 30 %.
La serie PRO, perfettamente compatibile con la standard, ha una potenza ed una sensibilità in ricezione maggiorata, caratteristiche che le permettono di essere impiegata in comunicazioni a lungo raggio; si distingue per la scritta PRO sull'involucro e la lunghezza leggermente superiore a quella degli altri moduli.
La serie 2 dei moduli XBee implementa completamente lo standard ZigBee; ciò aumenta la possibilità di realizzare reti complesse autoconfiguranti con funzioni di auto-routing, auto-riparanti e reti mesh. L'aspetto esterno è invariato ma viene aggiunta la sigla "serie 2" e si perde la compatibilità con i moduli della serie 1. La serie 2 introduce la possibilità di configurare un modulo come router, che, inserito in una rete, si comporta da ponte tra il Coordinator e gli End Device, oppure con altri router permette di estendere dinamicamente la rete. Anche in questo caso esiste la versione PRO con potenza maggiorata (la **Tabella 3** riepiloga le caratteristiche tecniche).
Vediamo ora, qui di seguito, gli elementi di base che formano una rete ZigBee.
• **Node**; questo termine indica un qualsiasi dispositivo della rete che consente lo scambio di dati via radio ed è identificato da un numero (Identify Device).

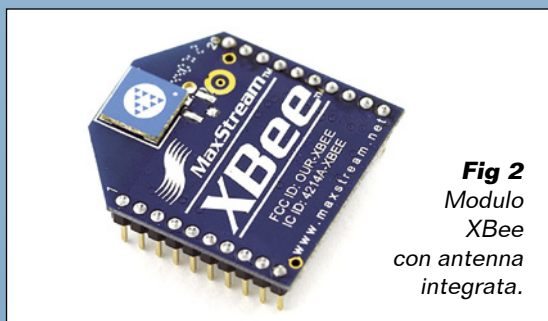



Fig 2
Modulo XBee con antenna integrata.

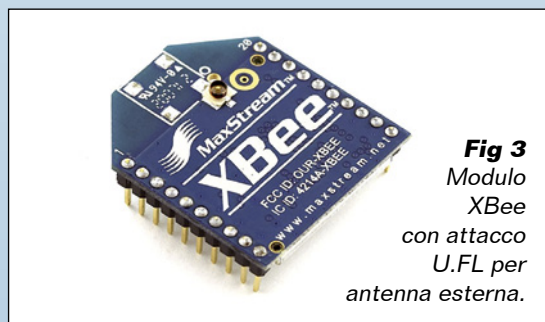



Fig 3
Modulo XBee con attacco U.FL per antenna esterna.

Fig 4
Modulo XBee con antenna esterna integrata.

Fig 5
Modulo XBee PRO con antenna integrata.

Fig 6
Modulo XBee PRO con attacco U.FL per antenna esterna.

Fig 7
Modulo XBee PRO con antenna esterna integrata.

Fig 8 Modulo XBee Serie2 con antenna integrata.

Fig 9 Modulo XBee Serie2 con attacco per antenna esterna.

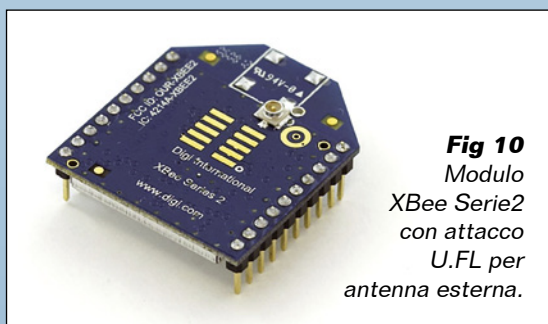



Fig 10 Modulo XBee Serie2 con attacco U.FL per antenna esterna.

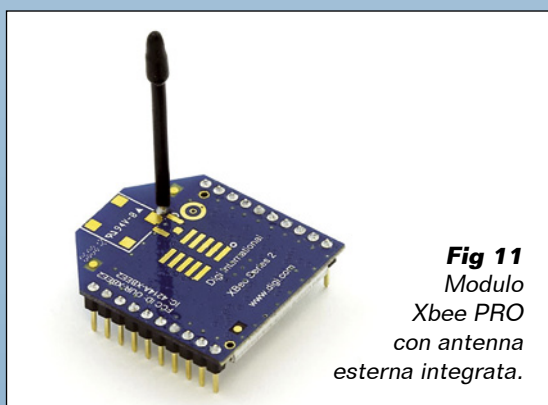



Fig 11 Modulo Xbee PRO con antenna esterna integrata.

• **Coordinator – ZC (Controller)**; rappresenta il nodo radice della rete ad albero, inizializza la rete, gestisce i vari nodi, opera la raccolta dati ed è in grado di operare da ponte verso altre reti; per ogni rete viene designato un Coordinator che contiene le informazioni sulla rete e le chiavi di sicurezza ed è in grado di configurare gli altri moduli della rete.
• **Router – ZR**; detto anche *Full function Device* (FFD) è un dispositivo client (genera informazioni e le invia al nodo centrale); può

Tabella 3 *- Caratteristiche tecniche moduli XBee serie 2.*

Parametro	XBee Serie2	XBee-PRO Serie2
Frequenza operativa	ISM 2,4 GHz	ISM 2,4 GHz
Potenza trasmissiva	2 mW	63 mW
Sensibilità ricezione	-92 dB	-100 dB
portata	40 m (indoor) 120 m (in aria libera)	90 m (indoor) 1.600 m (in aria libera)
consumo	TX 40 mA RX 40 mA Power-down < 1µA	TX 250 mA RX 55 mA Power-down < 4µA
Velocità di trasferimento RF	250 kbps	250 kbps
Velocità trasferimento UART	1,2 k ÷ 1M baud	1,2 k ÷ 1M baud
estensioni	Linee di A/D e digitali	Linee di A/D e digitali
Periferiche indirizzabili	65.000	65.000
Numero di canali disponibili	16	15
alimentazione	2,1 ÷ 3,6 V	2,7 ÷ 3,6 V

anche agire come intermediario per il rilascio verso il nodo centrale di informazioni da altri dispositivi.
• **End Device – ZED**; anche detto *Reduced Function Device*, RFID o semplicemente, nodo, è un dispositivo client: raccoglie informazioni e le invia al Coordinator o al Router, ma non opera come intermediario per altri dispositivi. È l'elemento più semplice ed economico, solitamente destinato ad oggetti quali interruttori, TV, radio, lampade, elettrodomestici, ecc. Normalmente fa parte di una sottorete a stella di tipo point-to-multipoint verso un Router o Coordinator.

RETI IMPLEMENTABILI CON I MODULI XBEE

Di seguito vediamo quali sono i tipi di rete wireless implementabili con i moduli XBee.

Rete point to point (PPP)
In questa rete due dispositivi dialogano tra di loro. Per migliorare l'immunità ai disturbi ed evitare conflitti che possano ritardare le comunicazioni, è possibile assegnare a questi due moduli un canale trasmissivo differente rispetto agli altri. La rete, anche se di soli due elementi, può contenere un master ed uno slave (**Fig. 12**).

Rete point to multipoint (PMP)
È una rete in cui un dispositivo dialoga con più dispositivi. La rete di tipo a stella ha una maggior complessità e impone di risolvere il problema dei conflitti, assegnando ad un solo dispositivo il compito di master mentre gli altri sono slave. Nel caso di reti ZigBee ci sarà un Coordinator e degli End Device (**Fig. 13**).

Rete peer to peer (P2P)
Si tratta di una rete definita paritaria, nella

quale non sono presenti dispositivi client o server, ma un numero di dispositivi che si equivalgono (in inglese *peer*) che svolgono le funzioni sia di server che di client. In questo modo ogni nodo è in grado di avviare e completare la comunicazione; è il caso di due portatili oppure di un PC ed uno smartphone che comunicano tra loro per scambiarsi file senza la necessità di utilizzare un Router. In ambiente Windows questa rete è anche denominata "rete ad hoc".

Rete Mesh
È la rete che impiega appieno le potenzialità del protocollo ZigBee, in cui sono presenti un Coordinator e diversi End Device supportati da Router utilizzati per espandere ed instradare le comunicazioni in modo dinamico. Questa rete può essere realizzata solo con i moduli della serie 2. Nelle reti mesh il dispositivo Coordinator è in grado di configurare dinamicamente gli altri moduli della rete; ogni dispositivo Router può svolgere operazioni di smistamento agganciandosi a sua volta ad altri Router. Espandendo in modo dinamico la rete, infatti, è possibile inserire "al volo" nuovi nodi lasciando che la rete stessa si autoconfiguri aumentando le distanze coperte. Nel caso la comunicazione con alcuni dispositivi si perda, si può recuperarla in automatico restaurandola tramite altri nodi; così la rete diventa molto robusta nei riguardi dei disturbi e dei guasti (**Fig.15**).
Le modalità di comunicazione implementabili con i moduli XBee sono tre e vengono elencate e descritte qui di seguito.

Transparent mode (pprz)
Questa modalità prevede che ogni modulo sia impostato come End Device e sia disabilitata l'associazione tra End Device in tutti i moduli; ogni dispositivo della rete dovrà avere gli stessi parametri ID e CH. Questa modalità è chiamata anche Transparent Mode e in essa due dispositivi sono utilizzati come radiomodem, comportandosi a tutti gli effetti come un normale cavo seriale: quello che si manda all'RX di un modulo arriva direttamente al TX dell'altro, senza dover programmare o settare nulla (tranne il baud-rate per la comunicazione); la packetizzazione dei dati e l'aggiunta di un checksum (per il controllo degli errori)

Confronto con i moduli a 433-868 MHz

Un'altra grande famiglia di moduli radio, disponibili sul mercato ed usati da molti appassionati per applicazioni wireless con i microcontrollori, utilizza la frequenza di trasmissione di 433 oppure 868 MHz. Rispetto agli XBee, questi offrono alcuni vantaggi tra i quali la tensione di alimentazione a 5 volt (direttamente compatibile con i microcontrollori) il costo leggermente inferiore ed una piedinatura a passo 2,54 mm standard, a differenza dei moduli XBee che, per limitare lo spazio, utilizzano un connettore a passo 2 mm. I moduli XBee hanno però alcuni indubbi vantaggi, come ad esempio la bidirezionalità di comunicazione, non sempre implementata sui moduli economici a 433 o 868 MHz. Altro vantaggio è la possibilità di indirizzare i moduli XBee in modo univoco, in quanto ogni modulo è già fornito di un numero seriale univoco, che permette a due moduli di dialogare solo tra di loro, ignorando i segnali di altri moduli nelle vicinanze. I moduli XBee implementano al loro interno anche una logica per la correzione degli errori ed una crittografia dei dati a 128 bit.

Fig 12 - Esempio di rete PPP.

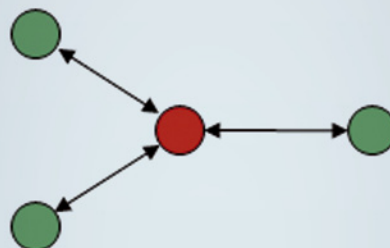
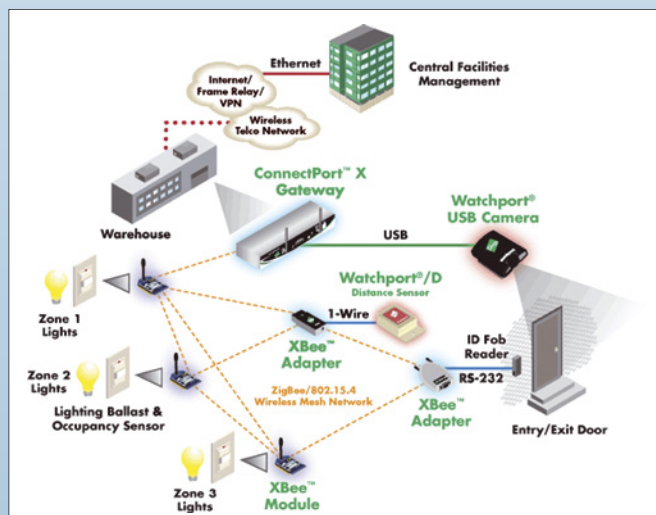



Fig 13 - Esempio di rete PMP.

Fig 14 - Esempio di rete P2P.

Fig 15 - Esempio di rete Mesh.

Fig 16 - *Esempio di rete Mesh per la gestione dell'impianto di illuminazione di un edificio di grandi dimensioni.*

devono esse fatte dall'utente prima dell'invio dei dati.

Command mode
Inviando i tre caratteri speciali "+++" il modulo XBee si porta in *Command Mode* e può essere configurato da semplici comandi AT, il che permette di modificare il baud-rate, l'indirizzo di destinazione ecc.

API mode.
Questa modalità è la più evoluta: oltre a poter configurare tutti i parametri di un modulo con i comandi AT (senza entrare in *Command Mode*) è possibile gestire le comunicazioni con gli altri moduli, ma anche ricevere notifica sui nodi associati, dissociati o resettati, richiedendo informazioni sugli indirizzi dei nodi della rete, oppure ricevere informazioni sullo stato di una trasmissione (buon esito) ed identificare l'indirizzo sorgente di una ricezione. In API *Command Mode* il modulo può essere configurato con maggiori possibilità di interazione con la rete e programmato per un funzionamento stand-alone (compresa la gestione delle linee di I/O e ADC) senza alcun supporto esterno. Ad esempio è possibile configurare un pin analogico affinché acquisisca ad intervalli regolari il segnale da un sensore ed invii i dati via radio. Con i comandi API è possibile configurare i moduli direttamente dall'applicazione Host (il microcontrollore, nel nostro caso).

Fig 17
Adattatore XBee logica 5 V.

Fig 18
Adattatore XBee logica 5V con modulo XBee inserito.

USO PRATICO
Come avrete intuito, i moduli XBee sono completi e progettati per essere utilizzati in reti di una certa complessità, il cui studio non sarà trattato in questa sede. Per le nostre applicazioni è sufficiente che un microcontrollore possa dialogare con un altro o con un PC in una piccola rete di tipo PAN, con al massimo qualche dispositivo. Il primo problema da affrontare riguarda la connessione hardware tra il modulo XBee ed il microcontrollore, in quanto i moduli Xbee operano a 3,3 V e adottano un connettore a passo 2 mm non proprio di facilissima reperibilità, quindi non possono essere utilizzati direttamente su di una basetta millefori sperimentale.
Per i collegamenti hardware tra microcontrollore e XBee è necessario considerare che la linea TX degli XBee può essere connessa direttamente alla RX del microcontrollore; pur giungendo una tensione di 3,3 volt a livello logico alto, il tutto funzionerà lo stesso. La linea TX del microcontrollore può essere connessa alla linea RX di XBee tramite un partitore di tensione che abbassi da 5 a 3,3 V la tensione, oppure interponendo un diodo con l'anodo rivolto verso il modulo Xbee (RX) e il catodo rivolto verso il microcontrollore (TX). È necessario siano abilitate le resistenze di pull-up del modulo XBee. In commercio esistono opportune interfacce già predisposte per queste funzioni, illustrate nelle figure 17 e 18.

Tabella 4 *- Piedinatura dei moduli XBee.*

Pin	Nome	Direction	Descrizione
1	VCC	-	Power supply
2	DOUT	Output	UART Data Out
3	DIN / CONFIG	Input	UART Data In
4	DIO12	Either	Digital I/O 12
5	RESET	Input	Module Reset (reset pulse must be at least 200 ns)
6	PWM0 / RSSI / DIO10	Either	PWM Output 0 / RX Signal Strength Indicator / Digital IO
7	PWM / DIO11	Either	Digital I/O 11
8	[reserved]	-	Do not connect
9	DTR / SLEEP_RQ/ DIO8	Either	Pin Sleep Control Line or Digital IO 8
10	GND	-	Ground
11	DIO4	Either	Digital I/O 4
12	CTS / DIO7	Either	Clear-to-Send Flow Control or Digital I/O 7
13	ON / SLEEP / DIO9	Output	Module Status Indicator or Digital I/O 9
14	[reserved]	-	Do not connect
15	Associate / DIO5	Either	Associated Indicator, Digital I/O 5
16	RTS / DIO6	Either	Request-to-Send Flow Control, Digital I/O 6
17	AD3 / DIO3	Either	Analog Input 3 or Digital I/O 3
18	AD2 / DIO2	Either	Analog Input 2 or Digital I/O 2
19	AD1 / DIO1	Either	Analog Input 1 or Digital I/O 1
20	AD0 / DIO0 / Commissioning Button	Either	Analog Input 0Digital IO 0 or Commissioning Button

Non vi dovete spaventare per la quantità di contatti di questi moduli, in quanto per l'utilizzo pratico sono richiesti solo VCC, GND, DOUT e DIN, oltre che limitate connessioni con supporto per aggiornamento firmware: VCC, GND, DIN, DOUT, RTS & DTR; ogni modulo include una resistenza di pull-up da 50 kohm sul pin di reset. Non è strettamente necessario comandare questo pin, quindi possiamo anche lasciarlo scollegato. Alcuni ingressi possono essere programmati con resistenza di pull-up o meno, mentre i pin non utilizzati vanno lasciati disconnessi.

CONFIGURAZIONE DEI MODULI

La configurazione dei moduli XBee può essere fatta tramite comandi AT, ma può diventare più semplice utilizzando il programma X-CTU, che la MaxStream-Digi fornisce gratuitamente per ambiente Windows. Questo software permette altresì di testare la rete radio tramite la misura dell'intensità del segnale e la qualità (errori rilevati) e consente anche l'upgrade del firmware dei moduli. X-CTU è scaricabile dal sito del produttore all'indirizzo *http://www.digi.com/support/*; selezionare come prodotto XCTU successivamente accedere alla pagina del download nella sezione "Diagnostics, Utilities and MIBs".
Per interfacciare il modulo XBee al PC è conveniente procurarsi un adattatore, disponibile sia in versione seriale che USB; quest'ultima è equipaggiata con il convertitore USB-seriale della FDTI, nel qual caso dovrete prima installare gli appositi driver scaricabili dal sito del produttore del chip *http://ftdichip.com/Drivers/VCP.htm*. Se state usando una scheda Arduino con interfaccia USB, questi driver li avete già installati.
In riferimento all'adattatore USB-XBee, i due LED denominati TX e RX permettono di monitorare il traffico dati del modulo XBee; un terzo è connesso alla linea RSSI di XBee ed un quarto indica presenza di alimentazione.
Inserite il modulo XBee sull'adattatore e collegatelo al PC; nel caso usiate la versione per USB, utilizzate *Gestione Risorse* del sistema operativo per sapere su quale COM è stato installato (**Fig. 25**). Avviate il software X-CTU e selezionate la cartella *PC_Setting*, quindi specificate la porta COM utilizzata dall'adattatore; se non compare nell'elenco cliccate su *User_Com_Port* e poi inserite manualmente il numero della porta sul campo *Com_Port_Number* e cliccate su *Add*. Adesso sarà disponibile nell'elenco delle COM (**Fig. 26**).
Per impostazione predefinita, i moduli comunicano con il protocollo 9.600 baud, 8 bit di dati, 1 bit di stop e nessun bit di parità; cliccate su *Test/Query* e lasciate che il software identifichi il modulo.
Se tutto è andato a buon fine, avrete la sigla del modulo e la versione del firmware (**Fig. 27**); ad esempio, inserendo un modulo XBee PRO, come risposta abbiamo ottenuto: *modem tipo XBP24 versione firmware 10E6* (modelli con firmware versione 1.x.
Selezionate la cartella *Modem_Configuration*

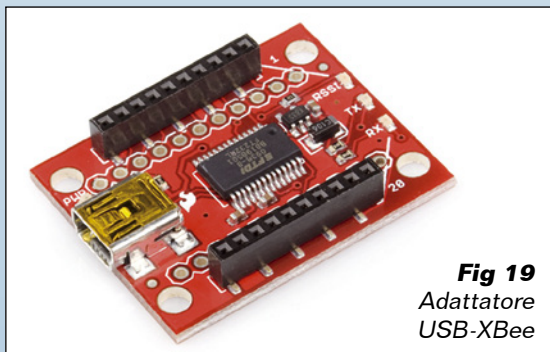



Fig 19
Adattatore USB-XBee

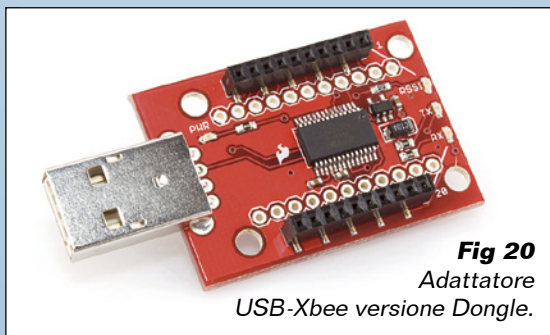



Fig 20
Adattatore USB-Xbee versione Dongle.

Fig 21
Adattatore USB-XBee V.Dongle installato

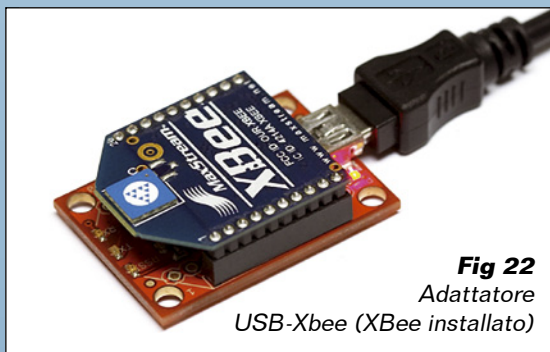



Fig 22
Adattatore USB-Xbee (XBee installato)

Fig 23
Adattatore RS232-XBee.

e cliccate su *Modem_Parameter - Read* per ottenere la configurazione del vostro modulo. Se il modulo è più recente del software sarà necessario aggiornare il data base delle funzioni, connettendovi ad internet e cliccando su *Download_New_Version* (**Fig. 28**).
Con il comando *Read* otterrete l'intera configurazione presente nella memoria del modulo (**Fig. 29**). Analizziamo di seguito, in dettaglio, i parametri fondamentali.

- CH è il canale di comunicazione (frequenza del modulo radio); solo i moduli con lo stesso canale possono comunicare fra loro, mentre moduli con un CH diverso creano di fatto una sottorete nella quale solo essi possono comunicare e coesistere con altre reti.
- ID è l'identificativo della PAN (Personal Area Network) sulla quale il modulo sta operando. Solo moduli con lo stesso ID e lo stesso CH possono comunicare tra di loro; in questo caso fanno parte della stessa PAN.
- DH-DL è l'indirizzo di destinazione dei messaggi. Impostando DH =0 e DL inferiore a 0xFFFF, i messaggi trasmessi da questo modulo sono ricevuti da tutti gli altri moduli della PAN che hanno parametro My uguale a DL. Se DH=0 e DL=0xFFFF la trasmissione di questo modulo sarà ricevuta da tutti i moduli. Se DH>0 e DL>0xFFFF la trasmissione di questo modulo verrà ricevuta esclusivamente dai moduli che hanno numero seriale uguale a questo SH (ricevente)=DH (trasmettitore) ed SL (ricevente)=DL (trasmettitore). Tutto questo è valido per moduli facenti parte della stessa PAN.
- SH-SL è l'identificativo (32 bit in totale) univoco impostato in fabbrica, ovvero ogni modulo ne ha uno diverso ed ovviamente non può essere modificato.
- My è l'indirizzo sorgente (il modulo in questione) da non confondersi con il numero di serie univoco per ogni modulo. Come vedremo questo indirizzo ci tornerà utile quando dovremo indirizzare delle trasmissioni solo verso alcuni moduli. Impostando MY=0xFFFF si disabilita la ricezione con indirizzo a 16 bit.
- RANGE è compreso tra 0 e 0XFFFF.
- PL è usato per impostare la potenza di trasmissione; a potenze inferiori corrispondono anche consumi inferiori.

Fig 24 Schema adattatore USB.

• BD: imposta il baud-rate per la comunicazione seriale. È importante conoscere sempre il valore impostato sul modulo, altrimenti non sarà possibile accedervi né per leggere la configurazione né per settarla. Per impostazione predefinita, sia i moduli che X-CTU sono impostati su 9.600 baud; questa è anche l'impostazione da utilizzare su Arduino per la comunicazione con XBee.

I moduli della serie 1 sono già settati per la comunicazione in *transparent mode*, operante in una tipologia di rete peer-to-peer in cui ogni modulo è utilizzato come End Device. Possiamo quindi utilizzare due moduli XBee (serie 1) per sostituire un collegamento cablato RS232 tra due dispositivi (transparent mode) senza la necessità di programmarli.
Utilizziamo quindi un secondo modulo XBee serie 1 (XBP24) che potrebbe essere inserito su un secondo adattatore ed interfacciato ad un secondo PC, oppure, come faremo noi, utilizzare il modulo XBee con una scheda Arduino in modo da realizzare un collegamento wireless PC-Arduino. Per connettere il modulo XBee alla scheda Arduino, potete utilizzare l'adattatore generico (**Fig. 17** e **Fig. 18**) realizzando i seguenti collegamenti:
• +5V Arduino al +5V adattatore XBee;
• GND Arduino -> GND adattatore XBee;
• TX Arduino -> RX Adattatore XBee;
• RX Arduino -> TX Adattatore XBee.

Ricordiamo che la linea TX di Arduino fa capo alla RX del convertitore FDTI, mentre la linea RX di Arduino fa capo alla TX del convertitore FDTI; ciò è un problema perché i due dispositivi XBee e FDTI inviano dati sulla stessa linea. L'alternativa è utilizzare l'apposita *XBee Shiedl*, che consente di realizzare tutti i collegamenti in modo semplice e veloce.
Tra le varie Shield disponibili nel mercato abbiamo usato la versione sviluppata in collaborazione con la Libelium, che fornisce supporto a questo indirizzo: *http://www.libelium.com/squidbee/index.php?title=Downloads*.

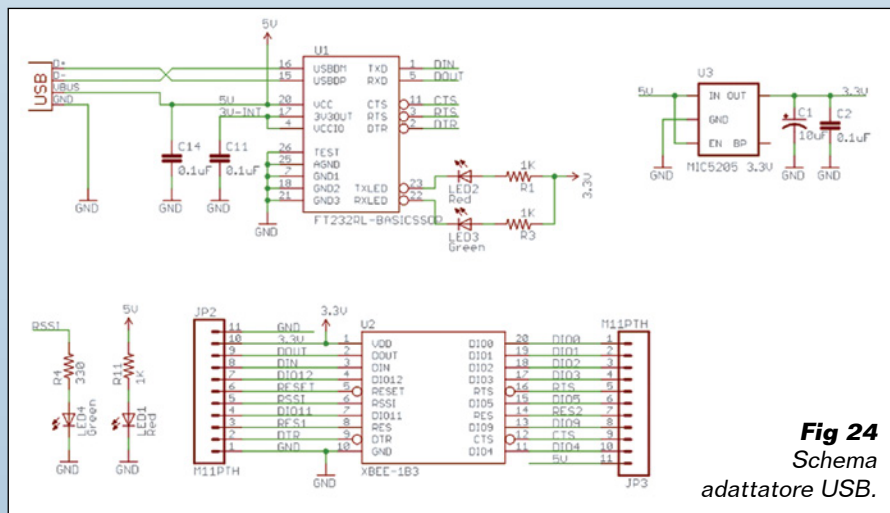



Fig 25 - Visualizzazione dispositivi seriali disponibili.

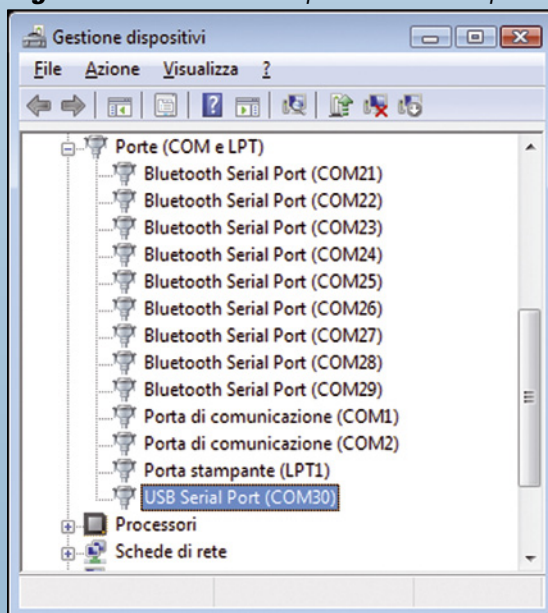



Fig 26 - Selezione COM sul software X-CTU.

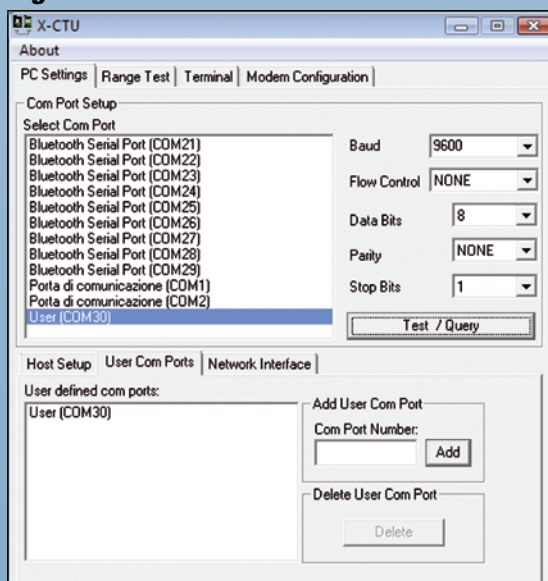



Fig 27 - Report della funzione Test/Query di X-CTU.

Fig 28 - Fase di scaricamento aggiornamento definizione moduli.

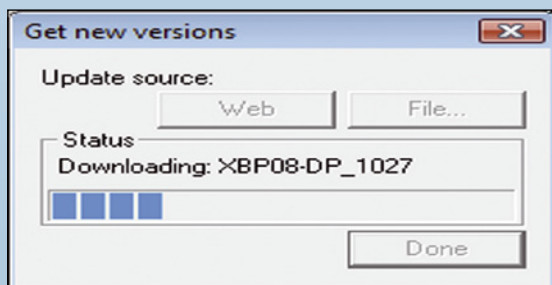



Fig 29 - Lettura parametri di default modulo XBee.

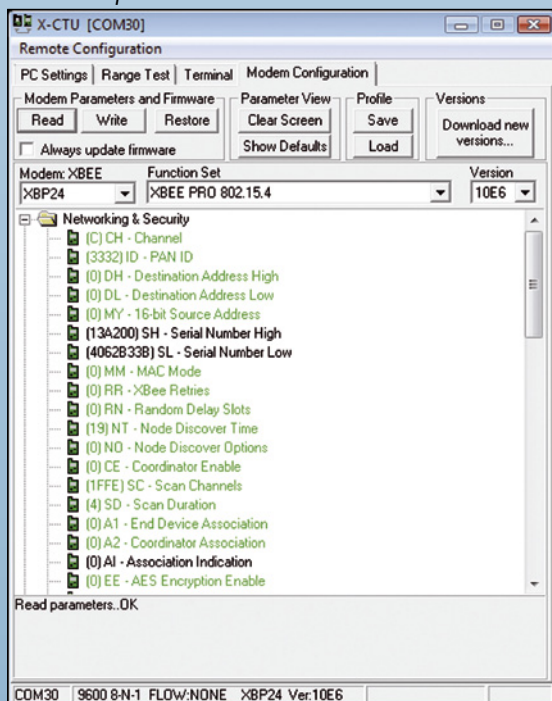



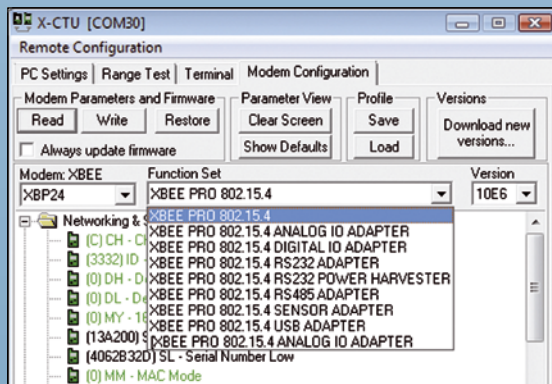
Fig 30 - Elenco opzioni del menu Function_Set di X-CTU.

La prima versione, siglata V1.0, faceva uso di soli componenti non SMD, mentre l'attuale versione V1.1, che abbiamo utilizzato (**Fig. 31**) dispone di piazzole idonee ad accettare i componenti sia classici che SMD.
Dallo schema (**Fig. 32**) si capisce che la tensione di alimentazione per il modulo XBee viene ricavata internamente da un apposito regolatore partendo dai +5 V della scheda Arduino. Le uniche linee utilizzate di Xbee sono la DIN (RX) e la DOUT (TX) le quali fanno capo a due jumper; la linea DIN ha un partitore per poter essere comandata con logica a 5 volt. I jumper permettono di configurare l'utilizzo della XBee Shield con il microprocessore Atmel oppure con la linea USB.
Con i jumper in XBee mode la linea DOUT di XBee è collegata al pin RX del microcontrollore, a sua volta connessa (in modo fisso) con la linea TX del chip FDTI. La linea DIN di XBee è connessa alla linea TX del microcontrollore, che a sua volta è collegata (in modo fisso) alla linea RX del chip FDTI. In questa modalità l'invio di dati da parte del microcontrollore avviene sia tramite il modulo XBee, sia tramite la USB. Il microcontrollore è abilitato a ricevere dati solo dal modulo XBee e non tramite la USB. Un eventuale dato in arrivo dalla USB e indirizzato al microcontrollore sarebbe in conflitto con la linea DOUT del modulo XBee. Usate questa modalità affinché il microcontrollore di Arduino usi il modulo XBee per trasmettere e ricevere dati.
Con i jumper in USB mode la linea DOUT dell'XBee è connessa alla RX dell'FDTI ed alla TX del microcontrollore, mentre la linea DIN dell'XBee è connessa alla TX dell'FDTI ed alla linea RX del microcontrollore. Il microcontrollore della scheda Arduino può quindi comunicare normalmente con il PC tramite la USB. Usate questa modalità per programmare via USB il microcontrollore. Rimuovendo il micro, il PC può comunicare direttamente con il modulo XBee tramite la USB, ma non disponendo delle linee RTS e DTR il collegamento non sarà equivalente all'utilizzo dell'adattatore USB-XBee. Ciò permette di usare il modulo XBee direttamente da PC, ad esempio per l'acquisizione di dati da una rete di sensori posti in luoghi diversi. Se il microcontrollore rimane inserito

Tabella 5 *- Lista dei principali parametri di default per modulo XBP24 (XBee serie1).*

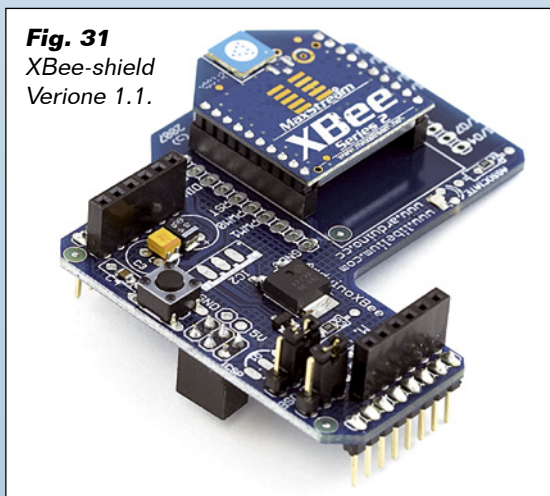
Parametro	default	funzione
CH	C	Set/read the channel number (Uses 802.15.4 channel numbers). RANGE:0XC-0X17
ID	3332	Set the PAN (Personal Area Network) ID. Set ID = 0xFFFF to send message to all PANs. RANGE:0-0XFFFF
DH	0	Set/read the upper 32 bits of the 64 bit destination address. Set the DH register to zero and DL less than 0xFFFF to transmit using a 16 bit address. 0x000000000000FFFF is the broadcast address for the PAN. RANGE:0-0XFFFFFFFF
DL	0	Set/read the lower 32 bits of the 64 bit destination address. Set the DH register to zero and DL less than 0xFFFF to transmit using a 16 bit address. 0x000000000000FFFF is the broadcast address for the PAN. RANGE:0-0XFFFFFFFF
MY	0	Set/read the 16-bit source address for the modem. Set MY = 0xFFFF to disable reception of packets with 16-bit addresses. 64-bit source address is the serial number and is always enabled. RANGE:0-0XFFFF
SH	13A200	Read high 32 bits of modems unique IEEE 64-bit source address. 64-bit source address is always enabled.
SL	4062B32D	Read low 32 bits of modems unique IEEE 64-bit source address. 64-bit source address is always enabled.

nella scheda esso potrà comunque comunicare via USB normalmente, ma né il computer né il microcontrollore possono comunicare con XBee.
Vogliamo ora implementare un semplice esempio, nel quale un dato inviato dal PC arriva alla scheda Arduino e questa risponde con lo stesso dato ricevuto: una specie di ECO che ci permette di verificare se il collegamento wireless funziona. Il **Listato 1** mostra le righe di codice corrispondenti. Il programma semplicemente aspetta un carattere in ricezione; il carattere "h" accende il LED della scheda, mentre il carattere "l" lo spegne; in ogni caso l'informazioine viene rispedita indietro sia tramite il modulo Xbee, sia tramite la USB. Se Arduino è connessa al PC ed è aperto l'Editor Arduino con attivato il tools Serial monitor (impostato su 9.600 baud) il carattere ricevuto sarà mostrato a video.

Listato 1

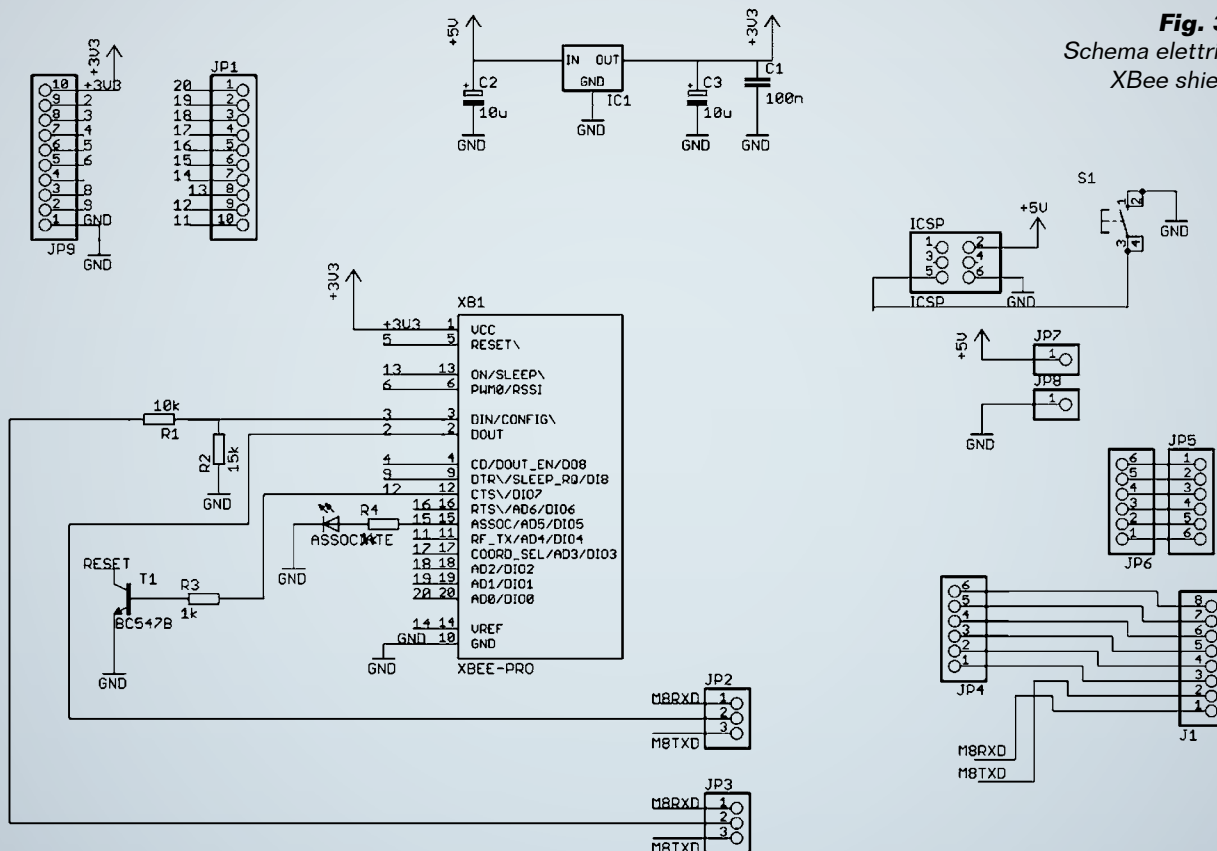
```
/*
  XBee_01
  Prova moduli Xbee
  riceve e rispedisce un carattere
  attiva LED 13 se riceve il carattere H
  spegne LED 13 de riceve il carattere L
*/

byte ChRX =0;    //carattere ricevuto
int Led = 13;       // LED connesso al pin digitale 13


void setup()
{
    pinMode(Led, OUTPUT);    // LED pin di uscita
    Serial.begin(9600);              // Velocità seriale = BD XBee!
}

void loop()
{
    while (Serial.available() > 0) {
        // aspetta arrivo di un carattere:
        ChRX = Serial.read();
        if (ChRX == 'h')
          digitalWrite(Led, HIGH); // sets the LED on

        if (ChRX=='l')
          digitalWrite(Led, LOW); // sets the LED off

        Serial.print(ChRX); //println
        delay(100);
     }
}
```

Connettiamo quindi il modulo Xbee alla Arduino Shield, che a sua volta sarà inserita nella scheda Arduino (nel nostro caso una Duemilanove); impostiamo i jumper su USB, connettiamo la scheda al PC e programmiamola normalmente (massima attenzione alla porta COM utilizzata). Spostiamo ora i jumper su XBee mode affinché il microcontrollore ATMega possa usare il modulo XBee.
Sulla XBee Shiled dovrebbe lampeggiare il LED *associated* per indicare l'associazione con il primo XBee. Sull'adattatore di quest'ultimo dovrebbe accendersi il LED RSSI (ad ogni dato ricevuto) ad indicare la presenza di segnale.
Avviamo X-CTU (che sarà sintonizzato sull'adattatore USB-XBee) e la schermata *Terminal*, ed iniziamo a digitare dei caratteri, uno alla volta con calma. Appena premiamo un tasto il carattere relativo viene inviato e, se tutto è a posto, lo stesso carattere torna indietro. Sull'IDE di Arduino vedrete, invece, i caratteri in arrivo sulla XBeeShield.
Sempre in X-CTU, andate sulla schermata *Range_Test* e in basso, alla voce *Create_Data*, inserite **1** e cliccate su *Create_Data*; viene impostato semplicemente il carattere **0** da spedire.
Cliccate ora su *Start* in modo che il carattere venga spedito in continuazione; il software verifica la corretta ricezione ed il relativo livello. Il perché inviamo un solo carattere è dovuto al fatto che lo sketch prevede, appunto, la ricezione e l'invio di un solo carattere; se

Fig. 31
XBee-shield Verione 1.1.

invece collegassimo RX e TX assieme sul secondo modulo (Arduino duemilanove e XBee shield omessi) potremmo trasmettere quanti caratteri vogliamo.
La linea RSSI programmata come tale (parametro P0=1) è un'uscita PWM con duty-cycle variabile a seconda del livello del segnale che viene valutato ad ogni ricezione di dati; con un piccolo filtro passa-basso si ottiene una tensione proporzionale alla potenza del segnale ricevuto. Il PWM rimane attivo per il tempo indicato dal parametro RP (in 100 ms).
Lo schema generale adottato è riportato nella **Fig. 37**; il secondo PC può essere omesso, essendo XBee gestito da Arduino, oppure è possibile utilizzare lo stesso PC facendo attenzione alle COM utilizzate.
Al posto di X-CTU, per inviare e ricevere dati con la scheda Arduino potreste usare anche il *Tools serial monitor* di Arduino. La scheda Arduino a sua volta potrebbe essere inserita in un robot, del quale volete avere il controllo da remoto, non solo per gestirne i movimenti ma anche per acquisire parametri sul campo.

PERSONALIZZAZIONE DELLA COMUNICAZIONE

Se utilizzassimo tre moduli con configurazione predefinita, il dato trasmesso da uno qualsiasi giungerebbe agli altri due.
Ora vediamo come si possono configurare diversamente i moduli: ad esempio, vogliamo comandare un robot a distanza, in modo che non vi siano interazioni con altri robot even-

Fig. 32
Schema elettrico XBee shield.

tualmente presenti nelle vicinanze che utilizzano la stessa tecnologia wireless; il modulo XBee A deve comunicare solo con il modulo XBee B. Sicuramente, per quanto detto prima entrambi i moduli dovranno avere stesso canale e stesso ID, cioè appartengono alla stessa PAN; dovranno, perciò, essere configurati in questo modo:

- Modulo A: $DL_A=SL_B$ $DH_A=SH_B$;
- Modulo B: $DL_B=SL_A$ $DH_B=SH_A$.

Avendo inserito come indirizzo di destinazione in ciascun modulo il numero seriale dell'altro, la comunicazione può avvenire solo tra questi due moduli. E se nessun altro conosce il numero seriale dei vostri moduli non potrà in alcun modo intromettersi nella comunicazione.

ESEMPI DI UTILIZZO

Vediamo adesso alcuni esempi di configurazione di sistemi wireless con moduli XBee; nel primo, il Modulo A può trasmettere solo a B, mentre quest'ultimo può trasmettere a chiunque. La configurazione è la seguente:

• Modulo A: $DL_A=SL_B$ $DH_A=SH_B$;
• Modulo B: $DL_B=0$ $DH_B=0$.

Il Modulo A usa come indirizzo di destinazione proprio il numero di serie del Modulo B, che quindi è l'unico a poter ricevere i messaggi di A. B invece ha come indirizzo di destinazione 0 e quindi trasmette a tutti.
Un secondo esempio che possiamo fare, prevede tre moduli A,B,C che comunicano in questo modo:
• A invia i dati a C;
• B invia i dati a C;
• C invia i dati sia ad A che a B.

Quindi A e B non comunicano direttamente tra di loro. La configurazione è la seguente:
• Modulo A: $DL_A=2$ $DH_A=0$ $MY_A=1$;
• Modulo B: $DL_B=2$ $DH_B=0$ $MY_B=1$;
• Modulo C: $DL_C=1$ $DH_C=0$ $MY_C=2$.

Il Modulo C ha come indirizzo di destinazione lo stesso valore dell'indirizzo sorgente dei moduli A e B, quindi ogni suo dato inviato giungerà sia ad A che a B. Sia A che B hanno

Fig. 35 - *Funzione Terminal di X-CTU.*

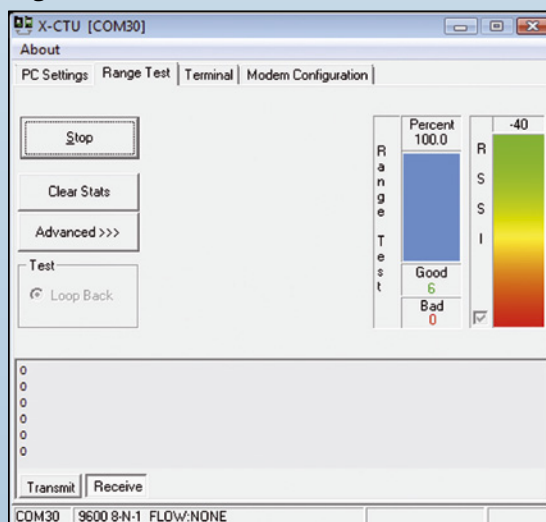



Fig. 36 - *Funzione Range Test di X-CTU.*

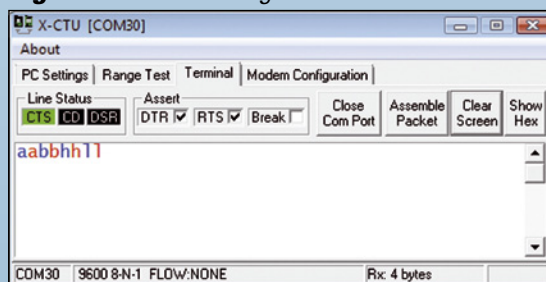
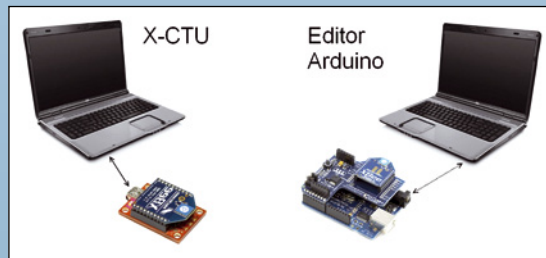



Fig. 37 - *Schema degli elementi utilizzati per l'esempio.*

come indirizzo di destinazione l'indirizzo sorgente di C (che è diverso da A e B) quindi solo a questo modulo giungeranno le loro trasmissioni.
Ovviamente se disponete di una sola postazione di programmazione dovrete programmare prima un modulo e successivamente l'altro, utilizzate il software X-CTU ed impostate i vari parametri, poi premete *Write* per scrivere nella memoria del microcontrollore i nuovi valori.

PROGRAMMAZIONE DEI MODULI XBEE SERIE 2.

La serie 1 non implementa la funzione di router e non è possibile configurare reti mesh. Con l'introduzione della serie 2, le modalità

Fig. 38 *- Avvio procedura di scrittura nuovi parametri su modulo.*

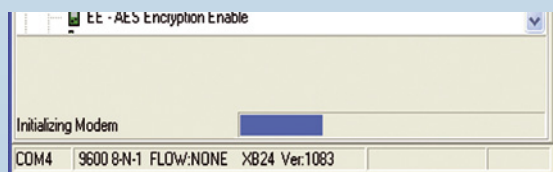



Fig. 39 *- Scrittura su modulo dei nuovi parametri.*

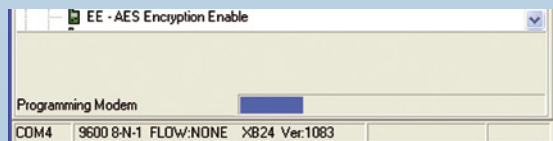



Fig. 40 *- Scrittura terminata con successo.*

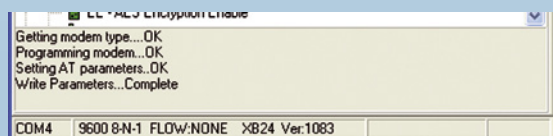
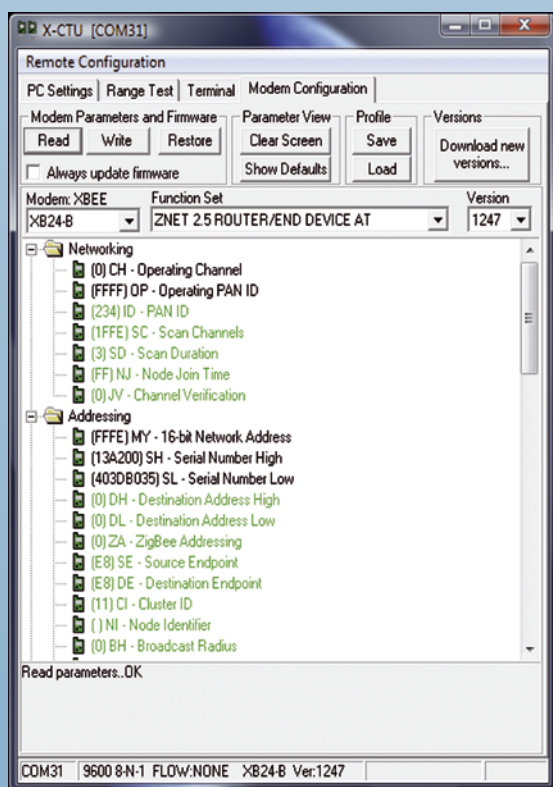
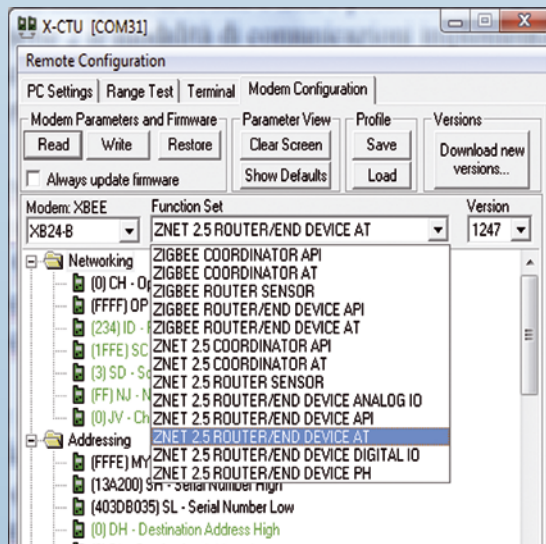



Fig. 41 *- Lettura dei parametri predefiniti di un modulo XBee serie 2.*

Fig.41 *- Possibili selezioni del menu Function_ Set per un modulo XBee serie 2.*

di comunicazioni implementabili sono diverse e dipendendo dalla versione firmware installata; non è quindi nota a priori la modalità operativa predefinita, né tantomeno è possibile sapere se supportano il transparent mode. Le versioni di firmware disponibili (*ZigBee OEM RF Modules by MaxStream - Digi International brand Firmware Versions*) sono:

- 1.0xx - Coordinator, Transparent Operation;
- 1.1xx - Coordinator, API Operation;
- 1.2xx - Router, End Device, Transparent Operation;
- 1.3xx - Router, End Device, API Operation.

Non è possibile utilizzare immediatamente i moduli, dato che prima è necessario programmarli; volendo nuovamente utilizzare due moduli per realizzare una semplice rete punto a punto in *Transparet Mode*, è necessario impostare un modulo come Coordinator e l'altro come End Device.
Come prima, dovete interfacciare il modulo al PC, in cui avvierete X-CTU; la lettura dei parametri predefiniti vi fornirà la schermata visibile nella **Fig. 41**. Noterete la sigla XB24-B, ad indicare la serie 2 e dal menu a tendina *Function_Set* si potranno leggere le varie impostazioni per il modulo; per impostazione predefinita, il canale trasmissivo è il D ed il PAN ID è 234. Due moduli identici, con le stesse impostazioni predefinite, non potranno comunicare tra di loro.
Lasciate un modulo impostato come ZNET 2.5 ROUTER/END DEVICE AT ed impostate l'altro come ZNET 2.5 COORDINATOR AT (**Fig. 42**). Per impostarlo è sufficiente selezionare questa dicitura sul menu a tendina e fare clic su *Write*. A questo punto il modulo End Device potrà inviare dati al modulo Coordinator, ma non viceversa.
Per permettere al Coordinator di inviare dati

all'EndDevice, possiamo ad esempio impostare il parametro DL=0xFFFF; questo abilita la trasmissione a tutti i moduli (broadcast mode) come visibile nella **Fig. 43**.
Se invece vogliamo che il Coordinator invii dati solo al nostro End Device, dovremo impostare:
- DHcoordinator=SHenddevice;
- DLcoordinator=SLenddevice.

Esiste anche una terza possibilità, che consiste nell'utilizzare entrambi i moduli come ZigBee ROUTER/END DEVICE AT (figure 44 e 45) con parametri:
- DHa=SHb;
- DLa=SLb;
- DHb=SHa;
- DLb=SLa.

Ora i moduli sono in grado di dialogare tra loro; in questa modalità i dati inviati sul TX di un modulo, arrivano direttamente sull'RX del modulo remoto e viceversa.

Fig.42 *- Impostazione di un modulo XBee serie2 come Coordinator.*

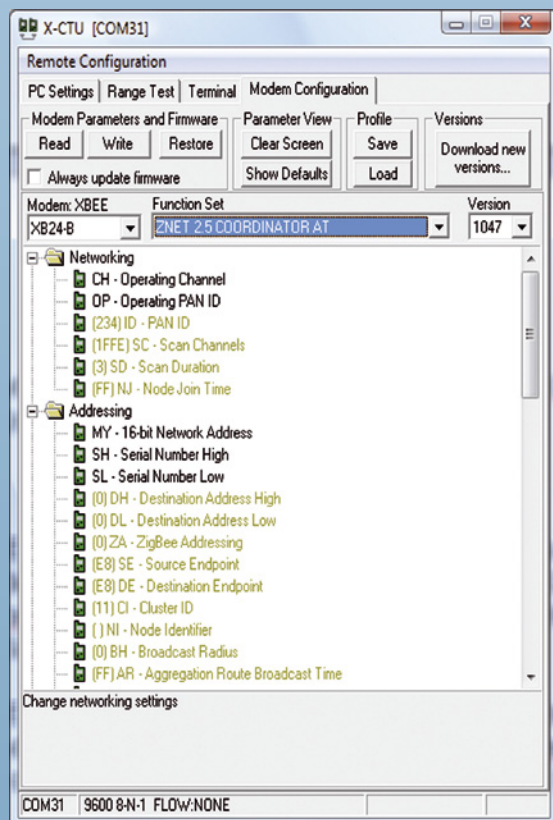



Fig. 43 *- Impostazione parametro DL modulo XBee serie 2.*

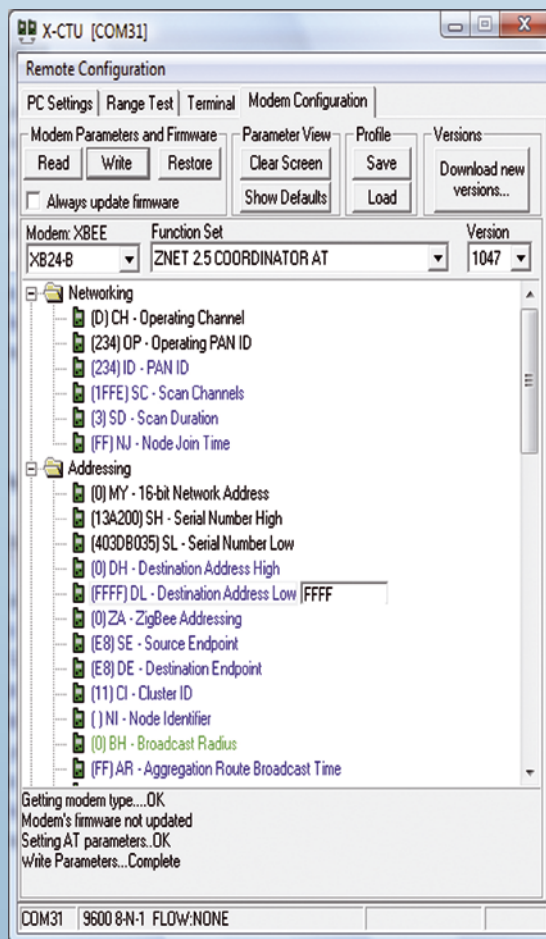



Fig. 44 *- Modulo XBee serie2 configurazione A.*

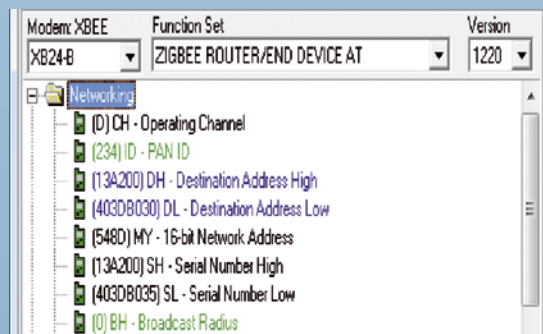



Fig. 45 *- Modulo XBee serie2 configurazione B.*

Conoscere e usare Arduino

dell'ing.
MIRCO
SEGATELLO

Impariamo ad utilizzare le SD-Card con Arduino, avvalendoci di uno shield realizzato appositamente. Settima puntata.

Ormai massicciamente utilizzate in fotocamere, palmari e navigatori satellitari per immagazzinare dati, le schede di memoria del tipo SD-Card sono diventate in breve tempo di largo utilizzo anche in apparecchiature elettroniche hobbistiche. Il perché è molto semplice: costano poco, hanno un'elevata capacità di memoria, sono di facile gestione e mantengono i dati anche in assenza di alimentazione. In breve, sono diventate il naturale complemento della risicata memoria di cui dispongono i microcontrollori, in tutte quelle applicazioni dove si renda necessario l'immagazzinamento o la lettura di una considerevole quantità di dati senza spendere grosse cifre: al momento in cui scriviamo, giusto per fare le nostre prove, abbiamo comperato una SD da 2 GB per appena 7 euro. Se consideriamo che un buon microcontrollore dispone di circa 1 kB di memoria non volatile, appare evidente l'abisso di prestazioni. La principale applicazione delle SD riguarda essenzialmente i data-logger, ovvero quelle apparecchiature che ad intervalli regolari memorizzano su di un supporto non volatile una serie di dati, come, ad esempio, parametri ambientali, posizioni rilevate da ricevitori GPS ed altro. Una SD-Card può essere facilmente gestita (lettura e scrittura) da qualsiasi PC tramite un economicissimo lettore, che nei computer più recenti è già integrato.
In questa puntata del corso vi insegneremo ad

***Tabella 1** - Caratteristiche delle schede di memoria.*

Tipo	dimensioni	Formattazione	Velocità	Capacità massima
MMC	30 x 23 x 1,4 mm			
RS-MMC	18 x 24 x 1,4 mm	FAT16	2 MB/s	512 MB
MMC mobile	14 x 11 x 1.1mm			
SD	32 x 24 x 2,1 mm			
Mini-SD	21,5 x 20 x 1,4 mm	FAT16 FAT32	2 MB/s	2 GB
Micro-SD (Trans Flash Card)	11 x 15 x 1 mm			
SDHC	32 x 24 x 2,1 mm			
Mini-SDHC	21.5 x 20 x 1.4 mm	FAT32	Classe 2 - 2 MB/s Classe 4 - 4 MB/s Classe 6 - 6 MB/s Classe 10 - 10 MB/s	32 GB
	11 x 15 x 1 mm			
SDXC	32 x 24 x 2,1 mm	FAT64	100 MB/s	2 TB

utilizzare le SD-Card con Arduino, ma prima di scendere nel dettaglio riteniamo sia doveroso spiegare qualcosa in più su questo genere di memorie.
La Multi Media Card (MMC) ha rappresentato uno dei primi standard di Memory Card: ha la stessa forma e la medesima piedinatura delle Secure Digital (SD), nate successivamente, con le quali è compatibile. Le SD-Card incorporano un microswitch che offre una protezione hardware contro la scrittura, ma ha portato lo spessore fino a 2,1 mm, quindi le schede MMC possono lavorare senza problemi in un socket per SD, mentre una SD non può entrare in un socket per MMC. Esiste anche una versione di MMC di dimensioni ridotte, chiamata RS-MMC (*Reduced Size MultiMedia Card*), del tutto identica nelle specifiche alla MMC classica. Attraverso un adattatore, che serve semplicemente ad adattare le dimensioni della card, le RS-MMC possono essere usate in qualsiasi slot per MMC (o SD).
Le Secure Digital (chiamate più brevemente SD) sono l'evoluzione delle MMC e raggiungono una capacità di 32 GB. Esistono tre formati (**Tabella 1**): le SD di dimensioni standard, le mini-SD e le microSD, chiamate anche TF Card (*Trans Flash Card*). Sia le microSD che le miniSD possono essere utilizzate con lettori di SD mediante semplici adattatori. Analogamente alle Multi Media Card, le SD utilizzano contatti superficiali anziché connettori maschio-femmina, fatto che ne aumenta ulteriormente la robustezza, ma le rende poco indicate per applicazioni in ambienti particolarmente gravosi, specie in presenza di vibrazioni.
Le SD superiori ai 2 GB e con una velocità minima di lettura/scrittura di 2,2 MB/s vengono chiamate SDHC (*Secure Digital High Capacity*) oppure SD 2.0, e non sono compatibili con i vecchi lettori di schede SD. Le SDHC (nate per accelerare la riproduzione dei contenuti multimediali) sono classificate in base alla velocità di trasferimento, detta *SD Speed Class Ratings* e definita dalla *SD Association*, che le divide in classi (**Tabella 1**). Le SD con capacità superiori ai 32 GB verranno chiamate con il nuovo termine: SDXC (*Secure Digital eXtended Capacity*) o SD 3.0. La capacità massima teorica dello standard SDXC è di 2.048 GB (2 TB) con una velocità del bus di 104 MB/s; il File System adottato è chiamato ExFAT (FAT64).
I vari lettori di schede SD non sono compati-

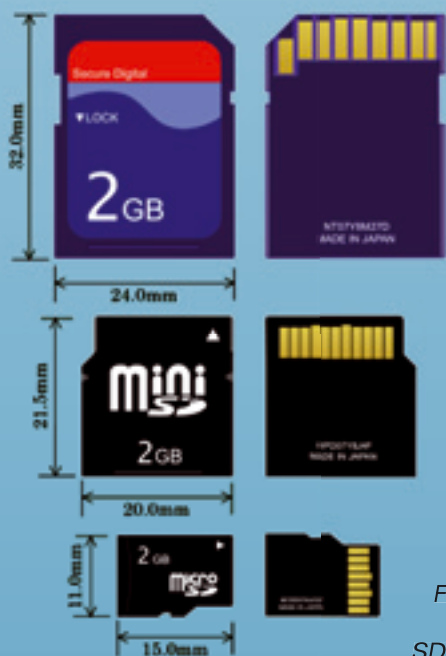



Fig 1 *Formati di una SD-Card.*

Fig 2 *Esempio di miniSD e relativo adattatore.*

***Tabella 2** - Contatti elettrici di una SD-Card (nella colonna Type, S=power supply; I=input; O=output con push-pull).*

	SD Mode			SPI Mode		
Pin	**Name**	**Type**	**Description**	**Name**	**Type**	**Description**
1	CD/DAT3	I,O	Card detection / Data Line 3 (Bit 3)	CS	I	Chip Select (Active low)
2	CMD	I,O	Command/Response	DataIn	I	Host to Card Commands and Data
3	VSS1	S	Supply Voltage Ground	VSS1	S	Supply Voltage Ground
4	VDD	S	Supply Voltage	VDD	S	Supply Voltage
5	CLK	I	Clock	CLK	I	Clock
6	VSS2	S	Supply Voltage Ground	VSS2	S	Supply Voltage Ground
7	DAT0	I,O	Data Line 0 (Bit 0)	DataOut	O	Card to Host Data and Status
8	DAT1	I,O	Data Line 1 (Bit 1)	RSV	-	Reserved
9	DAT2	I,O	Data Line 2 (Bit 2)	RSV	-	Reserved

***Tabella 3** - Contatti elettrici di una MM Card (Type: S=power supply; I=input; O=output).*

	MultiMediaCard Mode			SPI Mode		
Pin	**Name**	**Type**	**Description**	**Name**	**Type**	**Description**
1	RSV	NC	Not connected or Always	CS	I	Chip Select (Active low)
2	CMD	I,O,PP,OD	Command/Response	DataIn	I	Host-to-card Command and Data
3	VSS1	S	Supply Voltage Ground	VSS1	S	Supply Voltage Ground
4	VDD	S	Supply Voltage	VDD	S	Supply Voltage
5	CLK	I	Clock	CLK	I	Clock
6	VSS2	S	Supply Voltage Ground	VSS2	S	Supply Voltage Ground
7	DAT0	I,O,PP	Data0	DataOut	O	Card-to-host Data and Status

bili con le nuove versioni prodotte successivamente, ma solo con quelle precedenti; quindi un lettore di schede SDXC sarà compatibile con le schede SDXC, SDHC e SD, mentre un lettore di schede SD sarà compatibile solo con le schede SD. I dispositivi che non dichiarano il supporto SDHC non sono in grado di riconoscere le schede SDHC: ad esempio, schede da 4 GB non marcate come SDHC, non sono conformi né allo standard SD né a quello nuovo, SD2.0/SDHC.
Analizzando le caratteristiche elettriche di una SD Card, possiamo notare che l'interfaccia elettrica prevede 9 contatti ed una modalità di comunicazione compatibile con l'interfaccia parallela a 4 bit (SD Mode) e l'interfaccia seriale SPI (Serial Peripheral Interface) a due linee dati ed un clock (SPI Mode).
La tipica tensione di alimentazione è di 3,3 volt (i valori ammessi sono 2,7÷3,6 V) quindi i livelli dei segnali per la comunicazione dovranno essere analoghi. Se il microcontrollore cui è connessa la SD è alimentato a 3,3 V i segnali saranno applicati direttamente, mentre se si usa una logica a 5 V sarà necessario prevedere un adattamento tra i segnali, rispettando le seguenti indicazioni:
I segnali in uscita dalla SD-Card (3,3 V) possono essere applicati direttamente agli ingressi del microcontrollore, che pur alimentato a 5 volt riconoscerà correttamente il livello logico "uno" di 3,3 V, mentre i segnali in uscita dal microcontrollore (5 V) dovranno essere ridotti al livello di 3,3 V. Per fare questo esistono due soluzioni: utilizzare un partitore di tensione, in modo che i 5 V vengano ridotti a 3,3 volt, oppure usare dei traslatori di livello 5–›3,3 V come gli integrati 74HC4050N, 74AHC125N, 74LCX245. La differenza risiede nel fatto che

8 7 6 5 4 3 2 1 9

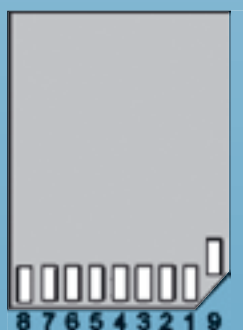



***Fig 3** Piedinatura di una SD Card.*

***Tabella 4** - Collegamento tra la SD-Card e Arduino.*

Pin SD	funzione	Pin morsettiera Arduino	funzione
1	CS	10	SS
2	DI	11	MOSI (Serial Data OUT)
3	VSS1	GND	GND
4	VDD	+3,3 V	+3,3 V
5	SCLK	13	Clock
6	VSS2	GND	GND
7	DO	12	MISO (Serial Data IN)
8	RSV	Non usato	
9	RSV	Non usato	

un partitore resistivo, pur essendo semplice ed economico, a causa delle capacità parassite non potrà gestire segnali ad elevata frequenza, mentre utilizzando un circuito elettronico traslatore (pure più costoso) consentirà di raggiungere velocità di comunicazione maggiori. L'interfacciamento tra una SD-Card ed un microcontrollore avviene tramite il bus seriale in SPI Mode, sfruttando essenzialmente due linee dati ed una linea di clock, oltre ad una linea per il Chip Select.

ARDUINO E LE SD-CARD

Bene, dopo questo "tutorial" sulle schede di memoria SD possiamo passare al nocciolo della questione, ossia a come usare le SD con i moduli Arduino.
Per facilitare l'utilizzo delle SD-Card con Arduino, in commercio sono reperibili diverse Shield con alloggiamento per SD-Card, oppure per microSD; le Shield vengono proposte con traslatore di livelli sia a partitore resistivo,

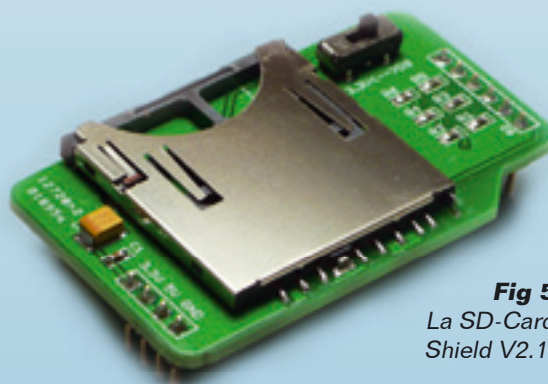



Fig 5
La SD-Card Shield V2.1.

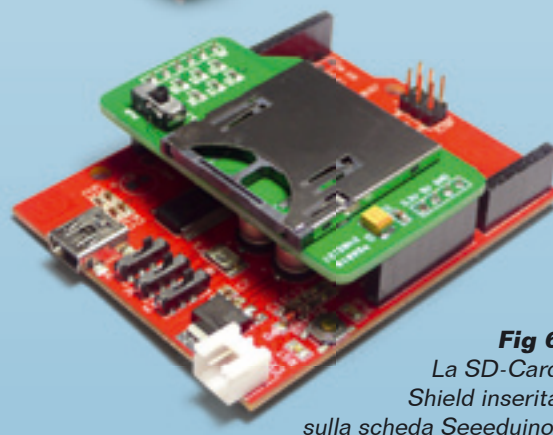



Fig 6
La SD-Card Shield inserita sulla scheda Seeeduino.

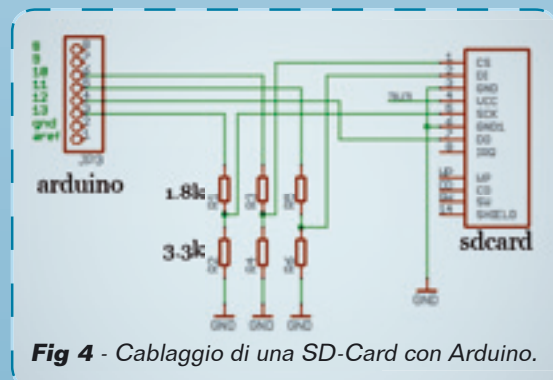



Fig 4 *- Cablaggio di una SD-Card con Arduino.*

Fig 7 *- Dettaglio inserimento SD-Card Shiled.*

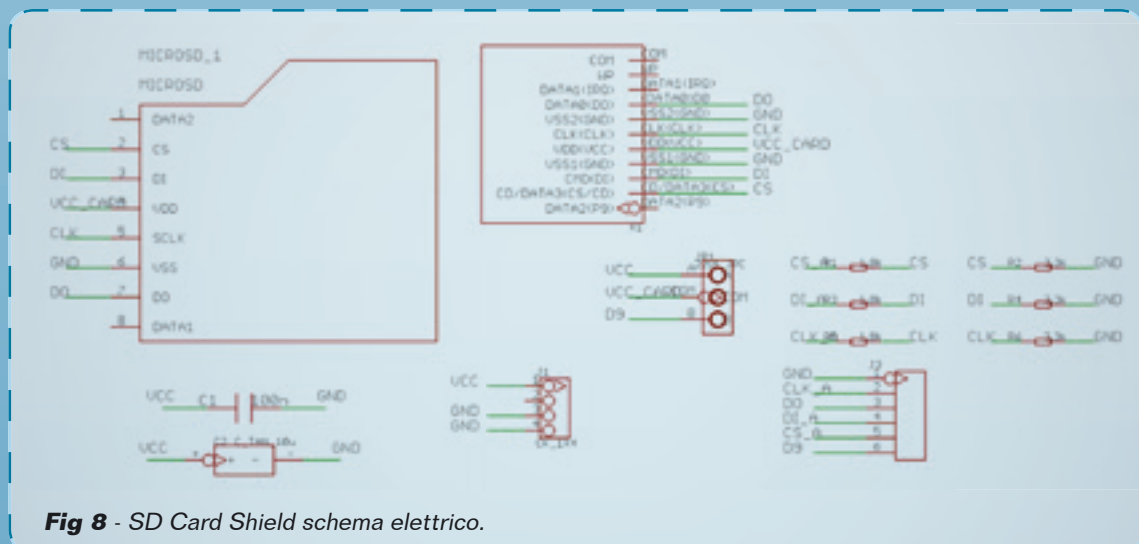
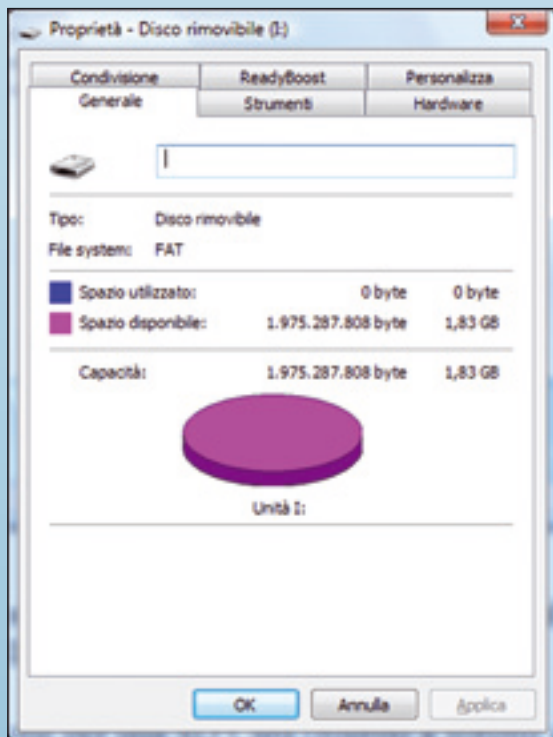



Fig 8 *- SD Card Shield schema elettrico.*

Fig 9 - Proprietà di una SD-Card da 2 GB visualizzate in Windows.

sia a logica integrata. Per la nostra applicazione abbiamo utilizzato la SD-Card Shield della SeeedStudio (*www.seeedstudio.com*) commercializzata dalla ditta Futura Elettronica (codice 7300-ARDUINOSDLETT). Questa unità, molto semplice e dal costo contenuto, permette il cablaggio veloce di una SD-Card standard con Arduino. Il piccolo deviatore presente sulla scheda permette di scegliere se prendere l'alimentazione per la SD-Card dal PIN9 o dai 3,3 volt di Arduino. A seconda della libreria utilizzata, saranno supportate MMC, SD o SDHC; come esempio per questo articolo abbiamo utilizzato una SD-Card standard da 2 GB. Se la inserite in un PC (tramite apposito lettore esterno, se non già integrato nel PC) e andate a visualizzare le proprietà della periferica (mostrata da Windows come *Disco Rimovibile*) troverete una schermata come quella visibile nella **Fig. 9**.
Oltre allo spazio disponibile, potrete leggere la formattazione presente, che nel nostro esempio è FAT, corrispondente ad una FAT16; sapere il tipo di FAT con cui è formattata una card è molto importante, perché non tutte le librerie di Arduino supportano lo standard FAT32 e quindi occorre accertarsene prima.

LA FORMATTAZIONE DELLE SD

La File Allocation Table (FAT) è un file-system primario per diversi sistemi operativi DOS e Microsoft Windows fino alla versione Windows ME. Windows NT e le successive versioni hanno introdotto l'NTFS e mantenuto la compatibilità con la FAT, così come molti altri sistemi operativi moderni (Unix, Linux, Mac OS).
La FAT è relativamente semplice ed è supportata da moltissimi sistemi operativi; queste caratteristiche la rendono adatta ad esempio per i floppy-disk e le Memory Card. Esistono varie versioni di questo file-system, in base a quanti bit sono allocati per numerare i cluster del disco: FAT12, FAT16, FAT32, ExFAT. L'intera memoria è suddivisa in aree denominate cluster (in Windows vengono chiamate *Unità di Allocazione*) composte a loro volta da un certo numero di bit. Quando un file viene salvato in memoria occuperà un certo numero (intero) di cluster. Cluster di grande dimensione permettono una migliore velocità di accesso ai file ma di contro offrono una scarsa efficienza nell'uso dei bit della memoria (frammentazione) perché, ad esempio, un semplice file di testo in cui vi è scritto un solo carattere occupa lo spazio di un byte, ma lo spazio occupato su disco sarà pari alla dimensione di un cluster.
La FAT in sé mantiene la traccia delle aree del disco disponibili e di quelle già usate dai file e dalle directory: la differenza fra FAT12, FAT16, FAT32, ExFAT consiste appunto in quanti bit sono allocati per numerare i cluster del disco. La prima versione del FAT fu la FAT12, un file-system per floppy-disk, i cui indirizzi per i cluster erano appunto a 12 bit e per questo poteva gestire dischi grandi al massimo 32 MB. Nel 1987 arrivò il formato che ora viene chiamato FAT16 e che ha i

Tabella 5 *- Tipo di formattazione per una SD-Card (riferito a sistemi operativi Windows).*

	ExFat (FAT64)	FAT32	FAT16	FAT12
Massima capacità	512 TB	32 GB	2 GB	16 MB
Massimo numero di file	2.796.202 (per directory)	4.194.304	65.536	
Massimo numero di cluster	2^64	2^32	2^16	2^12
Massima lunghezza nomi dei file	255	255	Standard - 8.3 estesa fino a 255	254

Tabella 6 - Struttura di un File system.

Area riservata			FAT		Root directory (solo FAT12/16)	Regione dati
Settore di avvio	Informazioni FS (solo FAT32)	Riservati (opzionale)	FAT N°1	FAT N°2		

cluster a 32 kB e 16 bit di indirizzamento; ciò fissò il limite massimo di una partizione in 2 GB. Per superare i limiti sulla dimensione dei volumi imposta dal FAT16, Microsoft decise di creare una nuova FAT chiamata FAT32, caratterizzata da indirizzi dei cluster a 32 bit, anche se in realtà si usano solo 28 bit.
Il file system FAT è strutturato in quattro sezioni diverse:

- la prima (area riservata) è il settore di avvio e contiene di codice del boot-loader per il sistema operativo;
- la regione FAT contiene la mappatura della regione dati e, per motivi di sicurezza, è in duplice copia;
- la Root Directory memorizza le cartelle ed i file presenti nella directory principale (chiamata root, appunto) ed è presente solo nella FAT12 e nella FAT16, mentre nella FAT32 la memorizzazione avviene direttamente nella regione dati;
- l'area dati, che è quella dove sono memorizzati i file (occupa la maggior parte della partizione).

Prima di iniziare ad utilizzare la nostra SD-Card con Arduino dobbiamo procurarci le relative librerie che ci assolvono da tutto il lavoro necessario ad implementare il protocollo di comunicazione; il bello di Arduino è proprio questo! La pagina web di riferimento di Arduino è la seguente: *www.arduino.cc/playground/Learning/SDMMC*.

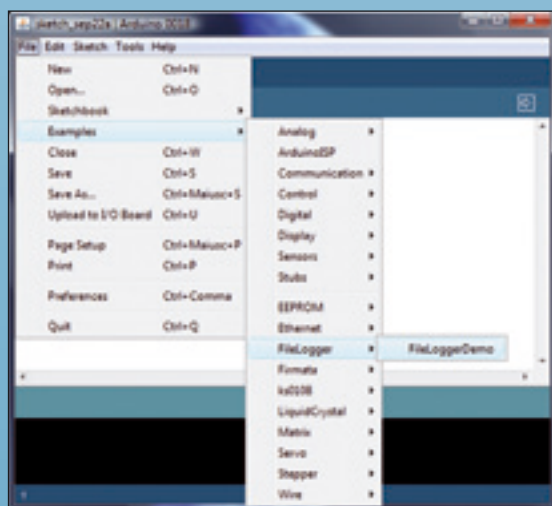



Fig 10 - IDE Arduino con libreria caricata.

Esistono diverse versioni di libreria adatte a gestire la FAT16 oppure la FAT32, alcune delle quali permettono la gestione del bus SPI ad alta velocità ma necessariamente l'adattatore per la SD-Card dovrà contemplare un traslatore di livelli a circuito integrato molto veloce. Ogni libreria, inoltre, utilizza proprie linee di controllo per SD-Card.
La prima libreria che andiamo a descrivere si chiama *SDuFAT*, sviluppata da David Cuartielles e disponibile all'indirizzo *http://blushingboy.net/p/SDuFAT/*, dove è reperibile nel file chiamato *SDuFAT.zip*.
La libreria SDuFAT è compatibile con le SD-Card standard formattate FAT16 con protocollo di comunicazione SPI e permette di leggere e scrivere su file già esistenti. Per poterla utilizzare, scompattate il file e copiate l'intera cartella SDuFAT nella cartella librarie del software Arduino; al primo avvio vi ritroverete con la disponibilità della libreria e relativo esempio.
Prendete la SD-Card, inseritela nel PC e verificate che sia formattata in modalità FAT; di solito quelle appena acquistate sono già formattate, ma potete tranquillamente riformat-

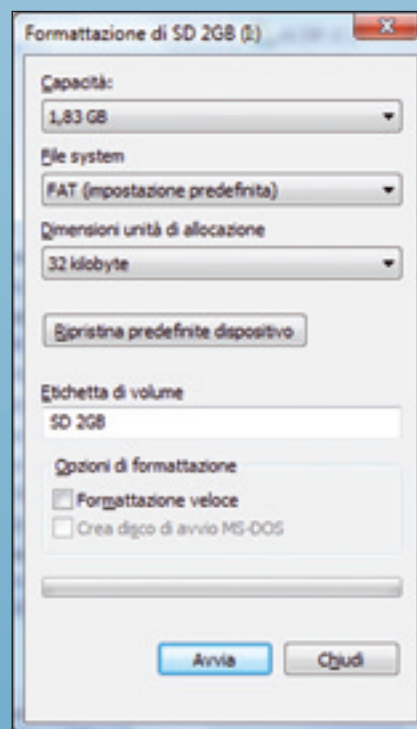
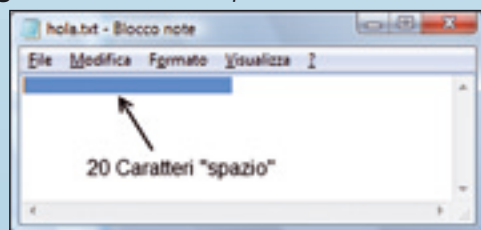



Fig 11 - Formattazione della SD-Card.

***Fig 12** - Il nostro file di prova, hola.txt.*

tarle, purché in modalità FAT e in un'unica partizione. Non è possibile gestire file se non nella Root della scheda.
Create all'interno della SD un "Documento di testo" di nome *hola.txt* e scrivete all'interno di questo file un certo numero di caratteri: ad esempio 20 caratteri "spazio". Attenzione: utilizzate un editor di testo che non aggiunga formattazione, come ad esempio "Blocco note" di Windows; in alternativa copiate, sulla SD-Card, il file di prova *hola.txt* presente nei file della libreria.
È necessario fare questo perché la libreria non è in grado di creare un file, né tantomeno un testo: semplicemente si occupa di scrivere o leggere all'interno dei caratteri che avrete preventivamente inserito.
Inserite la SD-Card nello slot della SD Shield, la quale a sua volta è inserita nella scheda Arduino; assicuratevi che il deviatore PWR sia in posizione D9, quindi connettetela via USB al PC, come al solito. Avviate l'IDE Arduino e caricate l'esempio SDuFAT.
Nel codice corrispondente, cercate la riga seguente:

```
// define the pin that powers up the SD card
#define MEM_PW 8
```

Quindi modificate il numero del pin di alimentazione in 9, con il comando seguente:

```
#define MEM_PW 9
```

Adesso, come al solito, caricate lo sketch su Arduino. Avviate Tools->Serial Monitor ed impostate un baud-rate di 19.200 bps. Dalle indicazioni riportate sullo sketch potete ricavare i comandi, via seriale, che gestiscono la SD-Card; per verificare che il collegamento funzioni, inserite il carattere 'H' (maiuscolo) e cliccate su SEND, Arduino risponde con l'elenco dei comandi (**Fig. 13**). Inviate il carattere 'L' per avere come report lo stato della SD Card.
Ora inviate il comando 'D', che permette di cancellare il file ed inserire in automatico il

***Fig 13** - Lettura dello stato del file hola.txt.*

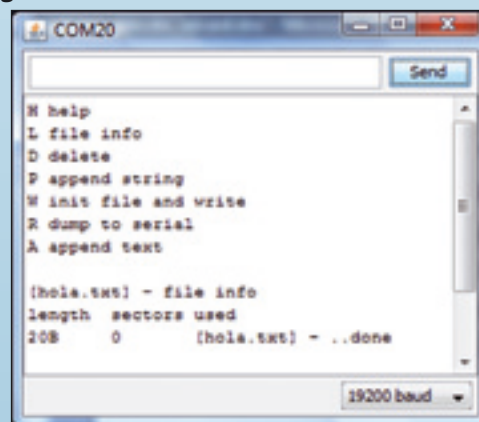
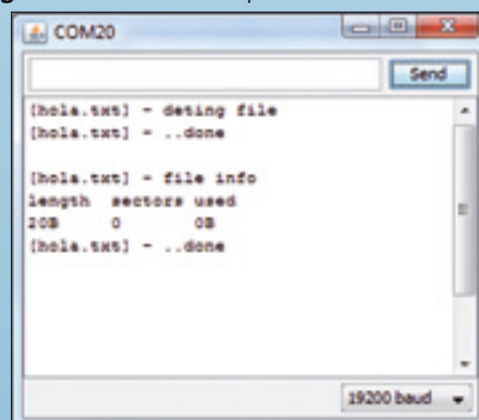



***Fig 14** - Stato del file dopo la cancellazione.*

carattere speciale (0x03 ASCII) di fine testo. Rileggete lo stato del file con il comando 'L'. Il parametro *length* vi riporta il numero di caratteri che avete inserito inizialmente; esso rappresenta anche il massimo numero di caratteri che potremmo scrivere. Il parametro *sectors* vi riporta il numero di settori occupati dal file, mentre *used* vi indica il numero di caratteri usati, che in questo caso è zero.
Inviate il comando 'P', che permette di scrivere il testo "hola caracola"; viene data conferma della scrittura con la dicitura "Done". I caratteri '\n' indicano l'inserimento di un "fine linea". Ripetete il comando 'L' per verificare lo stato del file, quindi inviate il comando 'R' per leggere il testo scritto nel file (**Fig. 15**).
Come potete vedere, ora il parametro *used* vale 15, ad indicare la scrittura di 15 caratteri.
Potete anche rimuovere la SD-Card da Arduino (senza necessariamente scollegare alcunché) ed inserirla nel PC per leggere il file.
Noterete la scritta "hola caracola" ed una serie di caratteri speciali aggiunti durante la cancellazione del file con il comando 'D'.
In totale, contando anche i caratteri "Null" (uno prima ed uno dopo il testo) troverete 20

caratteri come avete scritto all'inizio. Una seconda scrittura sullo stesso file comporterà la saturazione dovuta al superamento del numero massimo di caratteri.
Vediamo ora tutti i comandi (Basic) implementati in questa libreria:

- **ls(filename)**; riporta la dimensione massima, il numero di settori ed il numero di caratteri usati;
- **del(filename)**; cancella il file ed inserisce il carattere NULL (0x00 ASCII) all'inizio di ciascun settore;
- **print(filename, string)**; aggiunge una stringa alla fine del file;
- **println(filename, string)**; aggiunge una stringa e un EOL (End Of Line) a fine testo;
- **cat(filename)**; legge il contenuto del file o lo invia alla porta seriale;
- **append(filename)**; attende l'arrivo di testo dalla seriale e lo aggiunge a fine file.

Lo skecth prevede la gestione di questi comandi tramite l'invio di caratteri dalla seriale con il tool Serial monitor; più esattamente, le funzioni dei comandi sono le seguenti:

- **H** scrive l'elenco dei comandi;
- **L** scrive la lista delle informazioni del file ("hola.txt");
- **D** cancella il file("hola.txt");P: Scrive sul file "hola.txt" il testo "hola caracola";
- **W** inizializza il file inserendovi il testo ricevuto dalla seriale;
- **R** legge il contenuto del file o lo invia alla seriale;

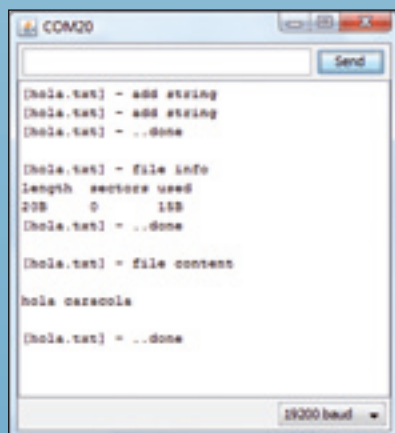



Fig 15 *- Scrittura sul file hola.txt.*

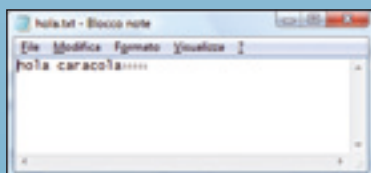



Fig 16 *- File hola.txt dopo l'operazione di Write.*

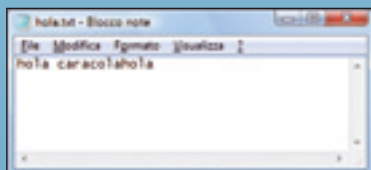



Fig 17 *- File hola.txt dopo seconda scrittura.*

Listato 1

```
case 'S':
  Sample=analogRead(0);
  result = SD.println("hola.txt", itoa(Sample, buf, 10));
  result = SD.println("hola.txt", "\r\n");
  Serial.print("AN0=");
  Serial.println(Sample);  // Report su serial monitor
  break;
```

- **A** aggiunge al file il testo ricevuto dalla seriale.

Considerando che uno dei principali utilizzi delle SD-Card nei circuiti elettronici è il salvataggio di dati nei data-logger, vi proponiamo anche un'estensione dello sketch di esempio, aggiungendo il comando 'S'. Con questo comando viene acquisito il valore dal pin analogico 0 e salvato sulla SD-Card. Il nuovo sketch si chiama *SD_01.pde* ed è disponibile assieme a tutti i file dell'articolo. Nel **Listato 1** riportiamo la parte di programma attinente al nuovo comando.
Oltre alla riga di acquisizione dall'ingresso analogico, viene usata la funzione ITOA per convertire il valore numerico in una stringa, necessaria per avere compatibilità con la funzione println della libreria SDuFAT. La seconda riga di println (caratteri "\r\n") consente di andare a capo e quindi di scrivere i valori tutti in colonna invece che nella stessa riga.
Dopo aver acquisito alcuni valori sarà possibile verificare la corretta scrittura leggendo la SD-Card con il PC (**Fig. 19**).
Esiste anche una versione semplificata della libreria SDuFAT, ottimizzata per ridurre la memoria occupata del microcontrollore ed aumentare la velocità di scrittura, denominata FileLogger e scaricabile dal sito Internet *http://code.google.com/p/arduino-filelogger/*; il file corrispondente si chiama *FileLogger.V0.X.zip* (dove X è la versione della libreria). Con questa libreria è possibile solo scrivere all'interno di una SD in FAT16, con il vantaggio che non è necessario preimpostare il testo al suo interno, ma è sufficiente che sia presente almeno un carattere; quelli scritti saranno accodati a questo. Diciamo subito che abbiamo testato con successo sia la versione 0.1 sia la 0.5, ma abbiamo riscontrato problemi con la versione 0.6. Dopo aver scompattato il file copiate la cartella FileLogger direttamente sulla

Fig 18 - Scrittura di dati numerici con esempio SD_01.pde.

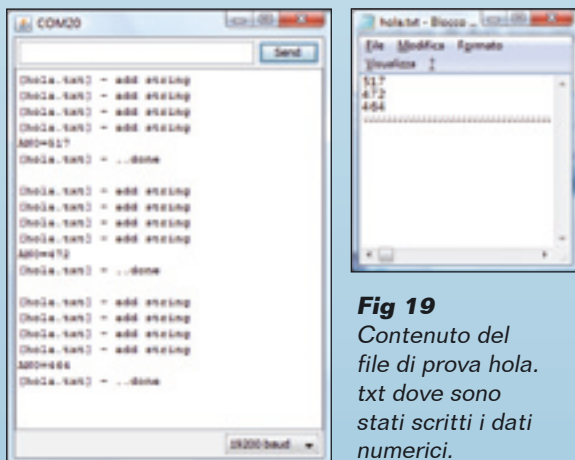



Fig 19 Contenuto del file di prova hola.txt dove sono stati scritti i dati numerici.

Listato 2

```
case 'S':
  Sample = analogRead(0);
  strcat(logStr,"\nVal= ");
  itoa(Sample,buffer,10);
  strcat(logStr,buffer);
  Serial.print (logStr);        // Eco su Serial monitor

  // conversione da array di char ad array di byte
  unsigned int length = (strlen(logStr)+1);
  byte bufSD[length];
  int i;
  for(i=0; i<length;i++)
  {
  bufSD[i] = logStr[i];
  }

  if( FileLogger::append("data.log", bufSD, 10) != 0) {
    Serial.println("Error write SD!");
  }
```

cartella libraries di Arduino. Il file di esempio si chiama FileLoggerDemo e prima di caricarlo sulla scheda Arduino dovete modificarne la riga che imposta il pin di alimentazione della SD-Card; fate ciò con le seguenti istruzioni:

// define the pin that powers up the SD card
#define MEM_PW 9

Assicuratevi che la SD Card sia formattata in FAT16, create al suo interno il file di nome "file.log" e scrivete in questo alcuni caratteri. Inserite la SD nello slot della shield ed avviate lo sketch; dovete aprire anche Serial Monitor (impostato sui 9.600 baud) per poter inviare il comando di scrittura "W".
Verrà scritto all'interno della SD Card il messaggio dichiarato con la riga #define MESSAGE; se l'operazione è andata a buon fine vedrete visualizzata la stringa 'OK' su serial monitor.
Abbiamo creato anche un esempio minimalista (SD_03.pde) nel quale oltre a scrivere un testo viene scritto il valore acquisito dall'ingresso analogico 0. Lo spezzone di codice che esegue questo passaggio è riportato nel **Listato 2**.
Come potete vedere le istruzioni sono molto semplici e prevedono essenzialmente l'acquisizione del dato dall'ingresso analogico 0 e la creazione della stringa 'logStr' che contiene il valore acquisito e che sarà visualizzata su Serial Monitor. La stringa formata da caratteri viene quindi convertita in un array di byte come richiesto dall'istruzione *FileLogger::append*, che si occupa di scriverla nella SD-Card.
Un'altra libreria, molto avanzata, con ottima documentazione riguardante anche la parte hardware, si chiama SDFATLIB ed è scaricabile dall'indirizzo web *http://code.google.com/p/sdfatlib/*; il file corrispondente si chiama *sdfatlib201000818.zip*. Questa libreria sfrutta il bus SPI alla velocità di 8 MHz, quindi il traslatore di livello deve essere a circuito integrato, tuttavia è stata testata sulla nostra Shield con successo. Permette di gestire scrittura, lettura, creazione e cancellazione di file e gestisce anche sottodirectory in SD e SDHC formattate FAT16 e FAT32. Scompattatene il file e copiate l'intera cartella *SdFat* nella cartella *libraries* di Arduino, così avrete a disposizione sia la libreria che molteplici esempi. In questo caso è necessario che il deviatore PWR sia in posizione 3,3 V. Aprite il file *Sd2PinMap.h* e verificate l'impostazione dei pin: SS_PIN deve essere configurato sul pin 10. La configurazione dovrà essere la seguente:

```
#else  // defined(__AVR_ATmega1280__)
// 168 and 328 Arduinos

// Two Wire (aka I2C) ports
uint8_t const SDA_PIN = 18;
uint8_t const SCL_PIN = 19;

// SPI port
uint8_t const SS_PIN = 10;
uint8_t const MOSI_PIN = 11;
uint8_t const MISO_PIN = 12;
uint8_t const SCK_PIN = 13;
```

A questo punto avviate Arduino, quindi aprite e caricate (UpLoad) l'esempio *SDFatWrite;* se tutto è andato a buon fine verrà creato il file *WRITE00.TXT*, con all'interno cento righe che riportano i millisecondi da quando è stato avviato lo sketch.
Giusto per testare le funzionalità di questa

Fig 20 - *Il file 'data.log" dopo la scrittura di 5 valori con sketch SD_03.pde.*

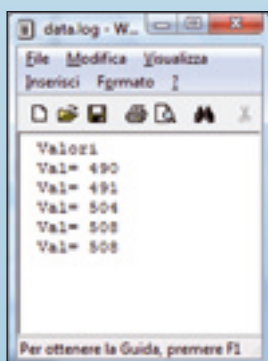
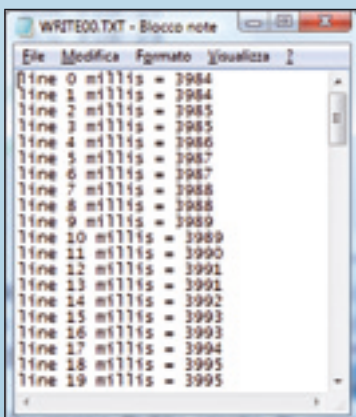



Fig 21 - *Esempio di file creato con libreria SDFAT.*

libreria, abbiamo scritto un piccolo sketch (*SD_02.pde*) che permette di aggiungere al file WRITE00.TXT 10 valori acquisiti dall'ingresso Analog0 ad intervalli di un secondo. Utilizzate il file già esistente WRITE00.TXT, dopo averlo cancellato, o createne uno di nuovo; non è necessario scrivere alcun contenuto.
La libreria FAT16LIB, è la versione semplificata per solo SD Card (no SDHC) formattate in FAT16, della libreria SDFATLIB.
Il file, che si chiama *fat16lib20100826.zip*, si scarica all'indirizzo *http://code.google.com/p/fat16lib/*. Verificate, aprendo il file SdCard.h, la corretta impostazione delle linee di controllo:

```
#else  // SPI pins
// pins for other Arduinos
/** Slave Select pin */
uint8_t const SPI_SS_PIN   = 10;
/** Master Out Slave In pin */
uint8_t const SPI_MOSI_PIN = 11;
/** Master In Slave Out pin */
uint8_t const SPI_MISO_PIN = 12;
/** Serial Clock */
uint8_t const SPI_SCK_PIN  = 13;
#endif  // SPI pins
```

All'interno del file *fat16lib20100826.zip* troverete anche molta documentazione utile per i collegamenti hardware.
Come indicato nella documentazione, è possibile (in caso di problemi) utilizzare il comando card.init(true) per inizializzare il BUS a 4 MHz invece di 8 MHz, come di default, per ottenere la compatibilità con traslatore di livelli a resistenze.
Questa libreria gestisce SD Card in FAT16 (non SDHC) con funzioni di lettura, scrittura, creazione e cancellazione di file nella sola directory principale con nomi in formato 8.3 (8 caratteri per il nome e 3 per l'estensione). È stata pensata per l'uso del BUS SPI ad alta velocità con traslatore di livelli a circuito integrato è tuttavia compatibile con la nostra SD Card Shield. Per il suo utilizzo fate riferimento alla libreria SDFATLIB.
Al fine di verificarne la compatibilità, abbiamo provato anche la libreria FAT16 creata da Ryan Owens e disponibile all'indirizzo *www.roland-riegel.de/ con file di nome sd-reader_source_20100110.zip*.
Questa libreria permette la scrittura e la lettura in file esistenti su MMC, SD e SDHC formattate FAT16 e FAT32 (FAT32 solo con ATmega328).
Anche in questo caso è necessario che il deviatore PWR sia in posizione 3,3 V e che nelle impostazioni generali sia impostato:

```
//Define the pin numbers
//#define CS    8
#define CS    10
#define MOSI    11
#define MISO    12
#define SCK    13
```

Per il resto, il principio di funzionamento è simile alle altre librerie descritte.
La libreria *SDFAT* scaricabile all'indirizzo *www.sparkfun.com/tutorial/microSD_Shield/SdFat.zip* con file di nome *SdFat.zip*, è la versione "Sparkfun" della libreria *SDFATLIB* dalla quale eredita tutte le caratteristiche.
Occorre tenere presente che più una libreria è complessa e ricca di funzioni, maggiori risorse richiederà in fatto di memoria; per questo motivo consigliamo di usare la libreria che abbia il minor numero possibile di funzioni richieste dalla vostra applicazione. A titolo di esempio, nella **Tabella 7** riportiamo l'occupazione in memoria di ciascuna libreria nel caso dell'esempio base.
Con ciò, abbiamo concluso questa puntata.

libreria	Binary Sketch Size
SDuFAT	8340 su 30720
FileLogger	6020 su 30720
SDFATLIB	11706 su 30720
FAT16LIB	8142 su 30720

Tabella 7 - *Memoria usata da ciascuna libreria.*

Conoscere e usare Arduino

dell'ing.
MIRCO
SEGATELLO

Analizziamo la neonata Arduino UNO, la scheda di sviluppo e prototipazione creata per sostituire la popolare Duemilanove.

Alla fine di settembre dell'anno scorso il team di Arduino ha annunciato alla stampa l'imminente disponibilità di due nuove schede, una delle quali è Arduino UNO; in questa puntata del corso riguardante il mondo Arduino vogliamo introdurvi ed insegnarvi ad utilizzare proprio questa novità. L'origine Italiana del prodotto è (come non mai) esaltata dal nome, il quale, non a caso, è UNO e non ONE (come si direbbe in inglese) e dalla presenza di un ben evidente marchio "Made in Italy" sulla confezione (e sul prodotto), con tanto di simbolo geografico dell'Italia. Arduino UNO è la prima su cui appare il nuovo logo scelto dal progetto Arduino, che è il simbolo matematico di infinito, scelto probabilmente per simboleggiare le infinite possibilità d'uso della scheda o la sconfinata espandibilità del progetto open-source da cui Arduino trae origine. Acquistando Arduino UNO, all'interno della confezione trovate alcuni adesivi ed un piccolo manuale -riportante le indicazioni principali sulla licenza di utilizzo- in lingua inglese, come è logico aspettarsi da un prodotto internazionale open-source. L'impressione che si ha, prendendo in

Fig 1
La confezione del nuovo Arduino.

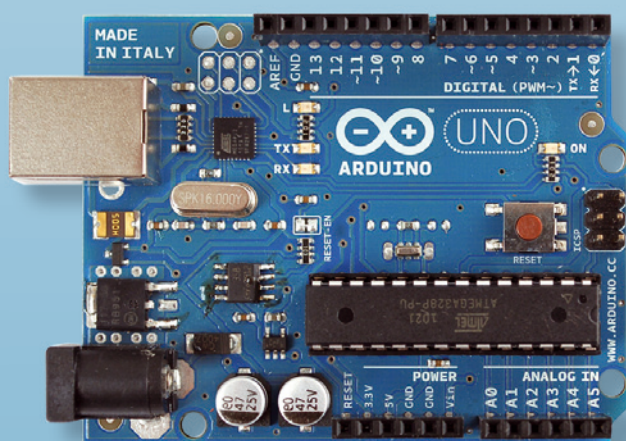



Fig 2
Aspetto dell'Arduino UNO vista dall'alto.

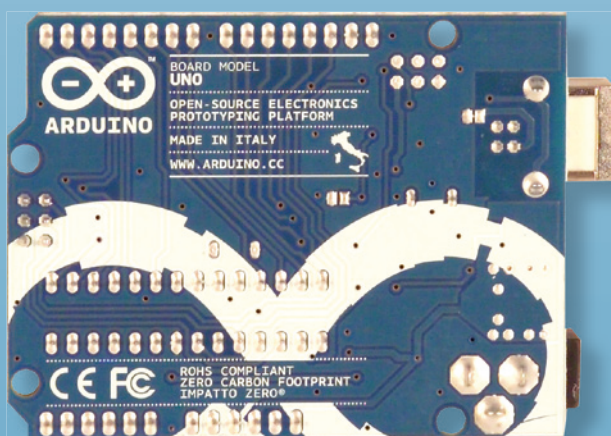



Fig 3
La scheda Arduino UNO vista da sotto.

mano questo nuovo hardware, è eccellente: la scheda è di buona qualità e lo stampato è di ottima fattura, con una serigrafia molto precisa e completa.

SCHEMA ELETTRICO

La particolarità dell'Arduino UNO e quindi ciò che la distingue nettamente da tutte le altre versioni proposte in questi anni, è che in essa per la connessione USB non viene più utilizzato il convertitore USB-Seriale della FDTI, bensì un microcontrollore ATmega8U2 programmato per funzionare come convertitore USB-Seriale. Questo nuovo prodotto della ATMEL è infatti un microcontrollore con a bordo un modulo Transceiver USB liberamente programmabile, come lo hanno ad esempio alcuni PIC della serie 18F della Microchip.
Il nuovo Arduino ha anche ricevuto la certificazione FCC sulle emissioni elettromagnetiche; le diciture "ROHS Compliant" e "Zero carbon footprint" fanno emergere l'interesse del team Arduino per la tutela dell'ambiente.
Arduino UNO è ancora basato sul microcontrollore ATmega328 (in formato DIP) e dispone di 14 pin di I/O (di cui 6 utilizzabili come uscite PWM), 6 ingressi analogici, un oscillatore a 16 MHz, un connettore per la programmazione In-Circuit ed un Plug per l'alimentazione. Come nelle ultime versioni di Arduino, è presente un connettore USB che, semplicemente connesso ad un Personal Computer, permette sia di alimentare la scheda, sia di programmarla.
Questa versione di Arduino è da intendersi come la 1.0 basata sulla nuova tecnologia per la connessione alla USB; la pagina di riferimento per comparare tutte le versioni hardware è la seguente: *http://arduino.cc/en/Main/Boards*. Al solito, il sito Internet di riferimento dove vedere le caratteristiche e trovare tutto quello che riguarda Arduino UNO è *www.arduino.cc* (in lingua inglese).
La **Tabella 1** riepiloga le caratteristiche più importanti della scheda Arduino UNO.
L'alimentazione della scheda può avvenire tramite la porta USB, ma è disponibile il solito connettore plug che accetta, in ingresso, una tensione non regolata con valore compreso tra 7 e 12 volt; in questo caso un semplice alimentatore non stabilizzato universale impostato sul valore di 9 volt è l'ideale, ma nulla vieta di

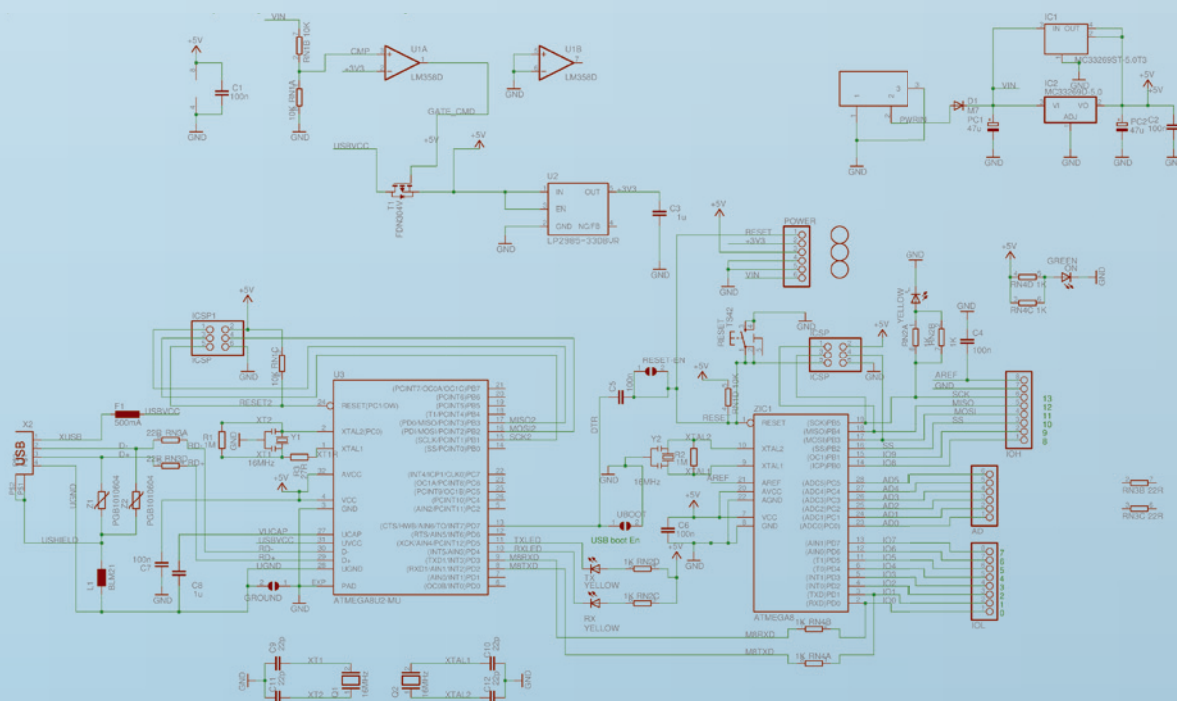
Tabella 1 *- Caratteristiche di Arduino UNO*

Microcontrollore	ATmega328
Tensione di lavoro	5 V
Alimentazione esterna (raccomandata)	7÷12 V
Alimentazione esterna (limiti)	6÷20 V
I/O digitali	14 (di cui 6 usabili come PWM output)
Ingressi analogici	6
Corrente per ogni I/O Pin	40 mA
Corrente prelevabile dal pin 3,3 V	50 mA
Flash memory	32 kB (ATmega328) di cui 0,5 kB usati per il bootloader
SRAM	2 kB (ATmega328)
EEPROM	1 kB (ATmega328)
Frequenza di clock	16 MHz

alimentare la scheda tramite una batteria a 9 o 12 volt. La sorgente di alimentazione viene riconosciuta in automatico e non serve impostare alcun deviatore. La porta USB è comunque protetta da accidentali cortocircuiti nella scheda e comunque da essa non vengono prelevati più dei 500 mA massimi ammessi.
I pin di alimentazione nella scheda Arduino sono i seguenti:

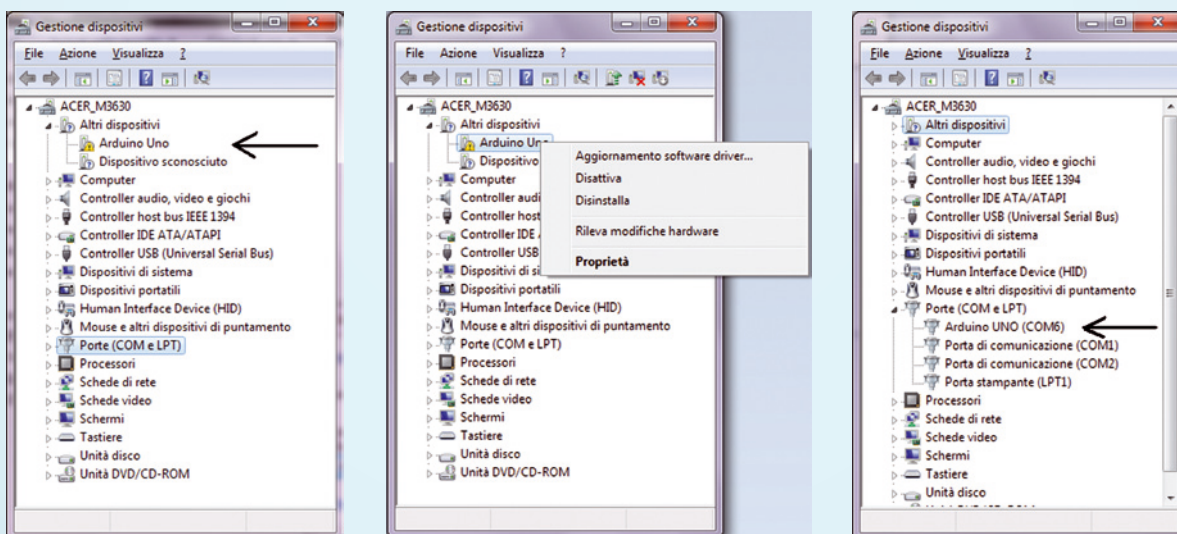
- VIN; questo, semplicemente replica la tensione fornita in ingresso sul connettore plug e può essere usato per alimentare altri circuiti che dispongano già di un regolatore di tensione (ad esempio gli shield applicati al modulo);
- 5 V; fornisce i 5 volt dello stabilizzatore di tensione interno alla scheda ed è utile per alimentare altri circuiti compatibili con i 5 volt;
- 3V3; questo pin fornisce i 3,3 volt ricavati dallo stabilizzatore interno alla scheda e consente di alimentare circuiti compatibili con tensioni di 3,3 volt (la massima corrente prelevabile è di 50 mA);
- GND; è il contatto di massa (GND).

CARATTERISTICHE DI ARDUINO UNO

Vediamo adesso le prerogative e la dotazione della nuova scheda Arduino, a partire dalla memoria: il microcontrollore ATmega328 dispone di 32 kB di memoria di programma, della quale 0,5 kB sono usati per il bootloader. Dispone inoltre di 2 kB di SRAM ed 1 kB di EEPROM utilizzabile, quest'ultima, per il sal-

Schema elettrico di Arduino UNO

vataggio di dati permanenti (mantiene i dati anche in assenza di alimentazione).
Quanto agli ingressi e alle uscite, ciascuno dei 14 pin di I/O può essere usato come pin di input o output e può erogare/assorbire una corrente massima di 40 mA; inoltre dispone di una resistenza di pull-UP del valore di 20÷50 kohm (attivabile tramite programmazione).
Arduino UNO dispone di sei ingressi analogici (A0,A1,A2,A3,A4 e A5) che per impostazione predefinita hanno risoluzione di 10 bit ed accettano una tensione compresa fra 0 e 5 volt; tuttavia è possibile usare l'ingresso Aref per modificare il range di misura.
Arduino UNO prevede degli interrupt esterni localizzati ai pin 2 e 3; questi possono essere configurati come trigger per eventi esterni, come ad esempio il rilevamento di un fronte di salita o di discesa di un segnale in ingresso.
Il modulo PWM del microcontrollore ATmega si può assegnare ai pin 3, 5, 6, 9, 10, e 11. Questi ultimi possono essere configurati via software per generare segnali PWM con risoluzione di 8 bit. Tramite un semplice filtro RC è possibile ottenere tensioni continue di valore variabile.
E veniamo alle porte di comunicazione: la seriale fa capo ai pin TX(1) e RX(0) che sono i corrispondenti dell'USART interno al microcontrollore e sono connessi al convertitore USB-Seriale della scheda.
Sempre in tema di comunicazione, va detto che il microcontrollore ATmega328 utilizza il modulo UART interno per comunicare, con livelli logici 0/5 volt, via seriale con altri dispositivi o con un PC. I segnali corrispondenti sono disponibili sui pin esterni (TX e RX) e risultano connessi anche al convertitore

Installare Arduino UNO su Seven

Seven è strutturato in modo da installare in automatico ogni periferica eventualmente cercando in rete i driver più opportuni. Arduino è però una periferica molto particolare, ragion per cui la procedura automatica potrebbe non andare a buon fine.
Quando inserite Arduino il sistema operativo identifica la presenza di una nuova periferica senza però riuscire ad installare i driver, e ovviamente non disponete del CD di installazione, in questo caso dovete procedere manualmente all'installazione dei driver. Dopo aver inserito la scheda Arduino Cliccate su Avvio-Pannello di controllo-Gestione dispositivi, troverete evidenziata la periferica Arduino con un punto esclamativo in quanto presente ma non correttamente installata.
Cliccate con il pulsante destro del mouse sopra la periferica e selezionate Aggiornamento software Driver... Quindi selezionate a mano la cartella "driver" del software Arduino, la procedura proseguirà adesso in automatico sino alla completa installazione.
A questo punto sarà presente la periferica Arduino correttamente installata e verrà indicato quale COM le è stata assegnata.

USB-Seriale della scheda, il che permette la comunicazione tramite la porta USB del computer. A differenza del chip della FDTI, per il quale era necessario installare appositi driver, con l'utilizzo dell'integrato ATmega8U2 ciò non è più necessario, in quanto vengono usati i driver comuni della periferica USB già disponibili con il sistema operativo. Tuttavia, con sistemi operativi Windows, per la corretta creazione di una porta COM virtuale è necessario installare un driver aggiuntivo. Allo scopo ricordiamo che Arduino UNO è compatibile con i sistemi operativi Windows, Mac OS X e Linux, per i quali sono previsti i driver.
Il bus SPI fa capo ai pin 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK) i quali possono essere programmati per realizzare una comunicazione SPI, appunto.
Arduino UNO prevede anche un bus I²C, localizzato ai piedini 4 (SDA) e 5 (SCL) che permettono di realizzare una comunicazione nello standard I²C a due fili, in abbinamento alla libreria Wire.
Concludiamo la carrellata sulle funzionalità di Arduino UNO con il piedino LED (pin 13) che è connesso a un LED interno alla scheda, utile per segnalazioni di diagnostica.
Infine abbiamo il Reset: questo contatto, portato a livello logico basso resetta il microcontrollore. La funzione corrispondente può essere attivata anche tramite il pulsante presente nella scheda Arduino.

INSTALLAZIONE ED UTILIZZO

Per il corretto utilizzo di questa nuova versione di Arduino è opportuno disporre dell'ultima versione del software -la 21- scaricabile gratuitamente all'indirizzo *http://arduino.cc/en/Main/Software*, la quale contiene sia i driver per Arduino UNO, sia i Driver per il chip FTDI di Arduino 2009. Con sistema operativo Windows, appena connettete Arduino al PC, esso verrà riconosciuto come nuovo hardware e vi verrà chiesta l'installazione del driver; allora non dovrete far altro che specificare come percorso la cartella "driver" del software Arduino (*Arduino UNO.inf*); il sistema operativo provvederà alla sua installazione ed alla creazione della porta COM virtuale. Fatto ciò avviate il software Arduino e specificate l'utilizzo di Arduino UNO, oltre alla COM alla quale è connesso. Il semplice esempio applicativo chiamato *File-Esempi-1.Basics-Blink* vi permetterà di testare il funzionamento della scheda. Inoltre sono disponibili tutorial e documentazione agli indirizzi web *http://arduino.cc/en/Tutorial/HomePage* ed *http://arduino.cc/en/Reference/HomePage*.

Fig 4 - *Impostazione del software per Arduino UNO.*

Grazie al Bootloader preinstallato a bordo, non è necessario utilizzare alcun programmatore esterno né è necessario rimuovere il microcontrollore, la connessione USB tra PC e Arduino è sufficiente a permettere la programmazione e la gestione della comunicazione. La funzione di autoreset interna alla scheda permette la programmazione con un solo click del mouse.
Su questa nuova versione di Arduino è presente un nuovo Bootloader basato sul protocollo STK-500, che occupa un quarto di memoria rispetto alla versione precedente. Adesso sono sufficienti 512 byte di memoria al posto dei 2 kb della precedente versione; inoltre, il nuovo bootloader può gestire la velocità di comunicazione sino a 115 kbps, contro i 57,6 k della precedente versione.
Tutta la documentazione sia hardware che software, compresi i sorgenti, è disponibile per il download sul sito di Arduino.
Arduino UNO è predisposta per comunicare in seriale con il PC semplicemente avviando il tools "Serial monitor" sull'ambiente di sviluppo; in questo modo un dato acquisito dalla scheda può essere facilmente visualizzato a video.
Il microcontrollore ATmega8U2 usato come convertitore USB-Seriale può essere facilmente programmato in quanto, al suo interno, è già precaricato il bootloader. In questo caso è possibile utilizzare le apposite piazzole di programmazione dopo aver attivato la modalità di programmazione saldando il piccolo jumper disponibile sul retro della scheda. Il software necessario per lo sviluppo dei pro-

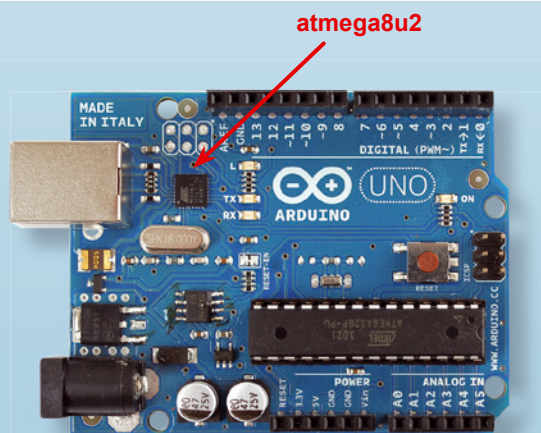



Fig 5 - *Posizione del microcontrollore 8U2 sulla scheda Arduino UNO.*

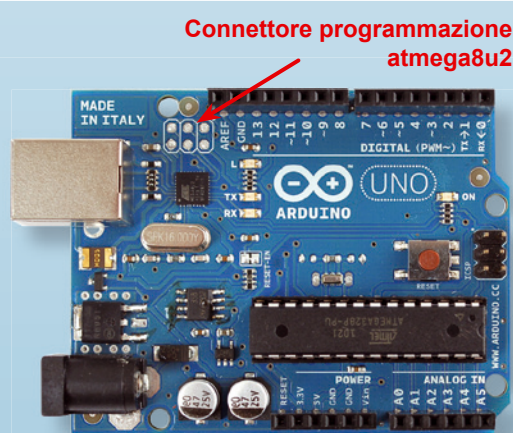



Fig 7 - *Piazzole per la programmazione di 8U2.*

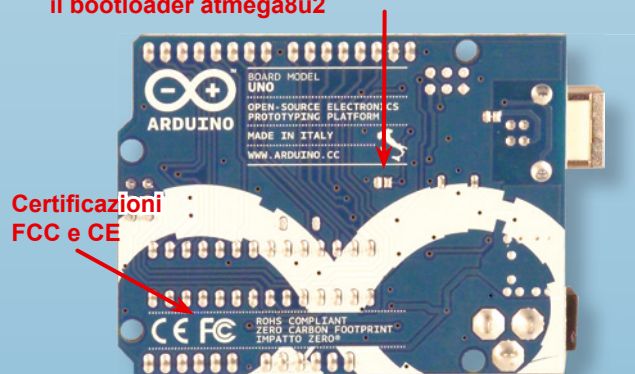



Fig 6 - *Jumper per attivare modalità aggiornamento firmware 8U2.*

grammi per l'ATmega8U2 si chiama Atmel's FLIP software per Windows (si scarica dalla pagina web *http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3886)* e DFU programmer per Mac OS X e Linux (si scarica da *http://dfu-programmer.sourceforge.net/*).
Ai più preparati non passerà inosservato il grosso vantaggio di avere il convertitore USB-Seriale programmabile; infatti sino ad ora Arduino poteva essere visto dal PC solo come una periferica seriale e infatti i driver del convertitore della FDTI creavano una seriale virtuale.
Adesso, invece, potendo programmate il convertitore, Arduino può essere visto dal sistema operativo del PC come una periferica ad-hoc. Nulla vieterebbe di far rilevare Arduino come una stampante e quindi, qualora inviaste il comando di stampa da un qualsiasi software, i dati giungerebbero ad Arduino, il quale, magari, può così controllare una macchina CNC! Allo stesso modo si potrebbe inventare un nuovo sistema di puntamento che, una volta connesso al PC, verrebbe riconosciuto come periferica tipo mouse ed il cursore sullo schermo si muoverebbe guidato da questa nuova periferica. Vi lasciamo immaginare quale interessante scenario Arduino UNO apre su quanto riguarda lo sviluppo di nuove periferiche per PC.

Conoscere e usare Arduino

dell'ing.
MIRCO
SEGATELLO

Interfacciamo Arduino con i ricevitori GPS utilizzando l'apposito GPS Shield e facciamo le nostre prime esperienze con la localizzazione da satellite. Nona puntata.

Questa volta ci occupiamo della interessantissima tecnologia GPS per la rilevazione della posizione sul globo terrestre (Global Position System). Come saprete sopra le nostre teste, a circa 20.000 km, sono posizionati dei satelliti che inviano costantemente sulla terra una serie di informazioni; queste, opportunamente captate ed elaborate, ci permettono di determinare la nostra posizione. Per ricevere ed analizzare i dati serve un GPS receiver completo di antenna, il quale, dopo essersi agganciato al segnale dei satelliti, fornisce sulla sua uscita una stringa di caratteri ASCII contenenti, oltre alla posizione, importanti dati. Il ricevitore GPS deve ovviamente essere nella condizione di ricevere segnali dallo spazio e quindi non funziona in ambienti chiusi come abitazioni o gallerie; inoltre, come tutti gli apparati radio operanti a microonde, riceve male in presenza di palazzi o costruzioni di grandi dimensioni, dato che a tali frequenze la propagazione del segnale è quasi in linea retta. Se la ricezione è buona, dopo l'accensione serve un tempo di acquisizione (denominato fixing) di solito compreso tra 30 secondi e 1 minuto, dopo cui il ricevitore passa alla fase di tracciamento, nella quale fornisce la posizione. Per chi vo-

Tabella 1 *- Contenuto della sentenza NMEA*

Sentenza NMEA	Descrizione
GGA	Dati relativi alla posizione
GLL	Posizione geografica
GSA	Elenco satelliti attivi
GSV	Satelliti acquisiti
RMC	Dati minimi essenziali sulla posizione
VTG	Direzione al suolo e velocità

lesse approfondire l'aspetto teorico di questa tecnologia suggeriamo di leggere "Tecnologia ed apparati GPS" – VISPA edizioni (lo potete richiedere alla nostra redazione o acquistare on-line dalla sezione *Libri* del nostro sito *www.elettronicain.it*).
La sequenza dei dati in uscita è composta da semplici caratteri ASCII codificati nella sentenza NMEA e contiene, oltre alla longitudine e latitudine del ricevitore anche l'altezza sul mare, la data e l'ora e le indicazioni dei satelliti agganciati. Oggigiorno il mercato fornisce svariati modelli di GPS facilmente utilizzabili nelle nostre applicazioni hobbystiche, a prezzi sempre più ridotti.
Per un acquisto consapevole ci sono alcuni parametri da tenere in considerazione: la dimensione, ad esempio, è un parametro importante soprattutto in apparecchiature che devono limitare gli ingombri; in questo rientra anche il discorso dell'antenna ricevente, che può essere integrata oppure esterna. Un altro parametro importante è il numero di letture che è in grado di fornire in un secondo: quasi tutti i modelli commerciali a basso costo forniscono una lettura al secondo, mentre i più evoluti arrivano anche a 10 letture al secondo.
L'alimentazione è un altro parametro di estrema importanza; di solito i ricevitori sono apparati progettati per funzionare in apparecchiature portatili e quindi pensati per essere alimentati a 3,3 o 5 volt con una corrente assorbita, variabile a seconda del modello, tra 10 e 50 mA, valore che va riducendosi con il migliorare della tecnologia.
Il numero dei canali gestibili è un altro parametro di solito sbandierato dai produttori, ma spesso una quantità inferiore di satelliti ben agganciati ed elaborati assicura migliori risultati di tanti satelliti ricevuti male.
La sensibilità fornisce un'indicazione della capacità di poter agganciare anche segnali deboli e quindi è indice di un buon funzionamento anche in zone disturbate o in cui giunge un segnale, debole come in centro città.
La precisione è un altro parametro di sicura importanza ed è strettamente legato al prezzo dell'apparato; tralasciando costose versioni per misure sul territorio (catasto o viabilità) i ricevitori commerciali si attestano su una precisione di ±10 metri o poco meno.
Per chi di voi si stia chiedendo a cosa possano servire i ricevitori GPS, diciamo che la loro principale applicazione è la memorizzazione di un percorso, nel caso di escursionisti, o la tracciabilità dei veicoli; più raramente, i GPS si usano per la navigazione di veicoli senza pilota. Molto più spesso li troviamo integrati negli smartphone di ultima generazione, così da rendere disponibile una serie di servizi tra cui la visita guidata alle città d'arte o da aiutarci a recuperare la strada di casa durante una vacanza in una nuova città; gli smartphone con ricevitore GPS possono diventare (con appositi software) navigatori satellitari dotati di assistenza vocale.
Tra i modelli disponibili sul mercato ne abbiamo selezionati un paio di facile reperibilità e pienamente compatibili con la nostra scheda Arduino; infatti, vista l'estrema miniaturizzazione di questi apparati, l'uso con una semplice breadboard e qualche spezzone di filo non risulta molto agevole.

Fig. 1
Aspetto del ricevitore GPS EM406A.

Fig. 2 *- Piedinatura EM406A.*

Tabella 2 *- Descrizione della piedinatura EM406A*

Pin	Nome	Funzione
1,5	GND	Zero volt alimentazione (ground)
2	Vin	Alimentazione positiva 4,5~6,5 V
3	RX	Segnale di ingresso RS232 per operazioni di programmazione
4	TX	Segnale di uscita RS232 (uscita dati)
6	1PPS	Fornisce un impulso ad ogni lettura (1 al secondo)

Il primo modello che prendiamo in considerazione si chiama EM406A ed è un modulo GPS a 20 canali con antenna integrata, le cui caratteristiche essenziali sono:

- chipset GPS a 20 canali SiRFStar III;
- alta sensibilità (-159 dBm);
- precisione nella posizione di ±10 metri, 2D RMS e di ±5 metri, 2D RMS;
- precisione nel calcolo della velocità di ±0,1 m/s;
- TTFF (Time To First Fix) estremamente efficace anche con livello di segnale non ottimale;
- supporta lo standard dati NMEA 0183 GGA, GSA, GSV, RMC, VTG e GLL;
- protocollo di comunicazione in logica TTL a 4.800 bps;
- tecnologia SuperCap che permette una rapida acquisizione dei dati dal satellite;
- antenna patch integrata;
- tempo di cold start di 42 s (da spento ad acceso);
- tempo di hot start di 1 secondo;
- alimentazione: 4,5 V ~ 6,5 V;
- consumo: 44 mA;
- dimensioni di 30x30x10,5 mm.

È possibile impostare il ricevitore per fornire diverse serie di dati in uscita; ad esempio, la sequenza GGA è quella illustrata nella **Tabella 3**. La stringa ricevuta, detta anche sentenza, è quindi la seguente:

```
$GPGGA,161229.487,3723.2475,N,12158.3416,W,1,07,1.0,9.0,M,,,,0000*18
```

Ma veniamo all'utilizzo pratico di questo ricevitore GPS (siglato EM-406A) reperibile presso la ditta Futura Elettronica (*www.futurashop.it*) con codice 8160-EM406A, completo di cavetto di collegamento. Per chi volesse utilizzarlo direttamente su di una breadboard per fare degli esperimenti, consigliamo di eliminare il connettore di uscita in dotazione e rimpiazzarlo con uno più adatto; se invece intendete realizzare da voi uno stampato per ricevere i segnali da questo modulo, potete trovare, sempre a catalogo Futura Elettronica, il relativo connettore (codice 7300-CONNEM406A).

***Tabella 3** - GGA data output.*

Nome	Esempio	Unità	Descrizione
Message ID	$GPGGA		GGA protocol header
UTC Time	161229.487		hhmmss.sss
Latitude	3723.2475		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12158.3416		dddmm.mmmm
E/W Indicator	W		E=east or W=west
Position Fix Indicator	1		0=no fix 1=fix OK (GPS SPS Mode) 2=fix OK (Differential GPS, SPS Mode) 4= fix OK (GPS PPS Mode, fix valid)
Satellites Used	07		Range 0 to 12
HDOP	1.0		Horizontal Dilution of Precision
MSL Altitude	9.0	meters	
Units	M	meters	
Geoid Separation		meters	
Units	M	meters	
Age of Diff. Corr.			Null fields when DGPS is not used
Diff. Ref. Station ID	0000		
Checksum	*18		
<CR><LF>			End of message termination

Per interfacciare correttamente questo modulo con la scheda Arduino, possiamo semplicemente utilizzare l'apposita GPS-Shield disponibile sempre dalla Futura Elettronica con il codice 7300-GPSSHIELD, fornita già montata ed equipaggiata con il connettore per il modulo EM406A; in essa sono presenti anche un pulsante di reset ed uno di ON/OFF per il modulo GPS. Non comprende, però, i connettori per il fissaggio alla scheda Arduino che dovrete acquistare a parte (7300-STRIP6 e 7300-STRIP8) e successivamente saldare. Questa scheda dispone inoltre di piazzole per il collegamento del modulo EM408, funzionante a 3,3 V. Per il modulo EM406, la scheda provvede a fornire l'alimentazione a 5 volt ed il cablaggio dei due segnali TX ed RX del GPS con le linee D2 e D3 di Arduino. Particolare attenzione deve essere posta nell'uso del deviatore DLINE-UART. Con il deviatore in posizione UART il modulo GPS è connesso alle linee digitali 0 e 1, ovvero le linee TX ed RX del modulo UART di Arduino. Con il deviatore in posizione

***Fig. 3** - GPS shield con modulo EM406A.*

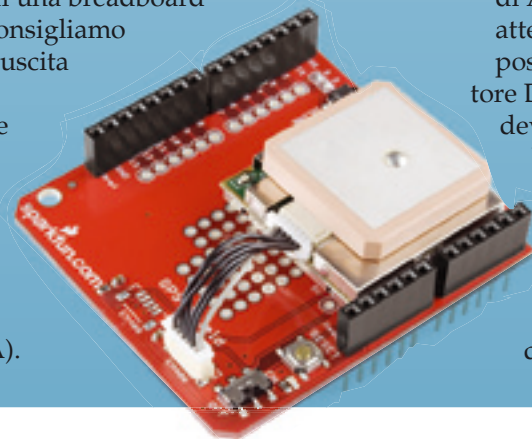
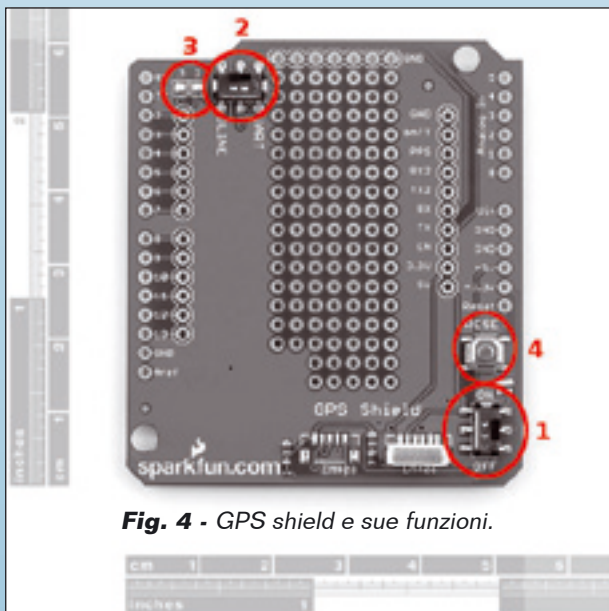



***Fig. 4** - GPS shield e sue funzioni.*

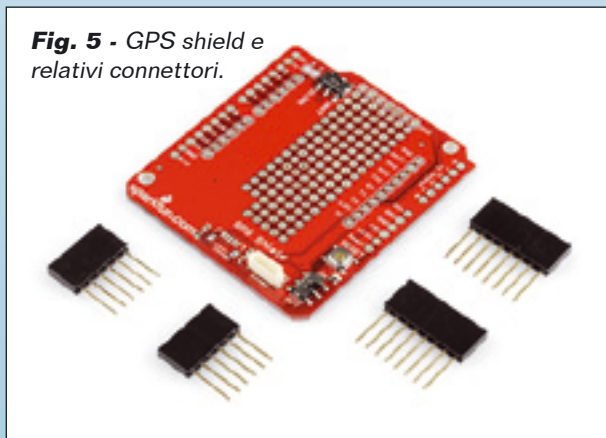
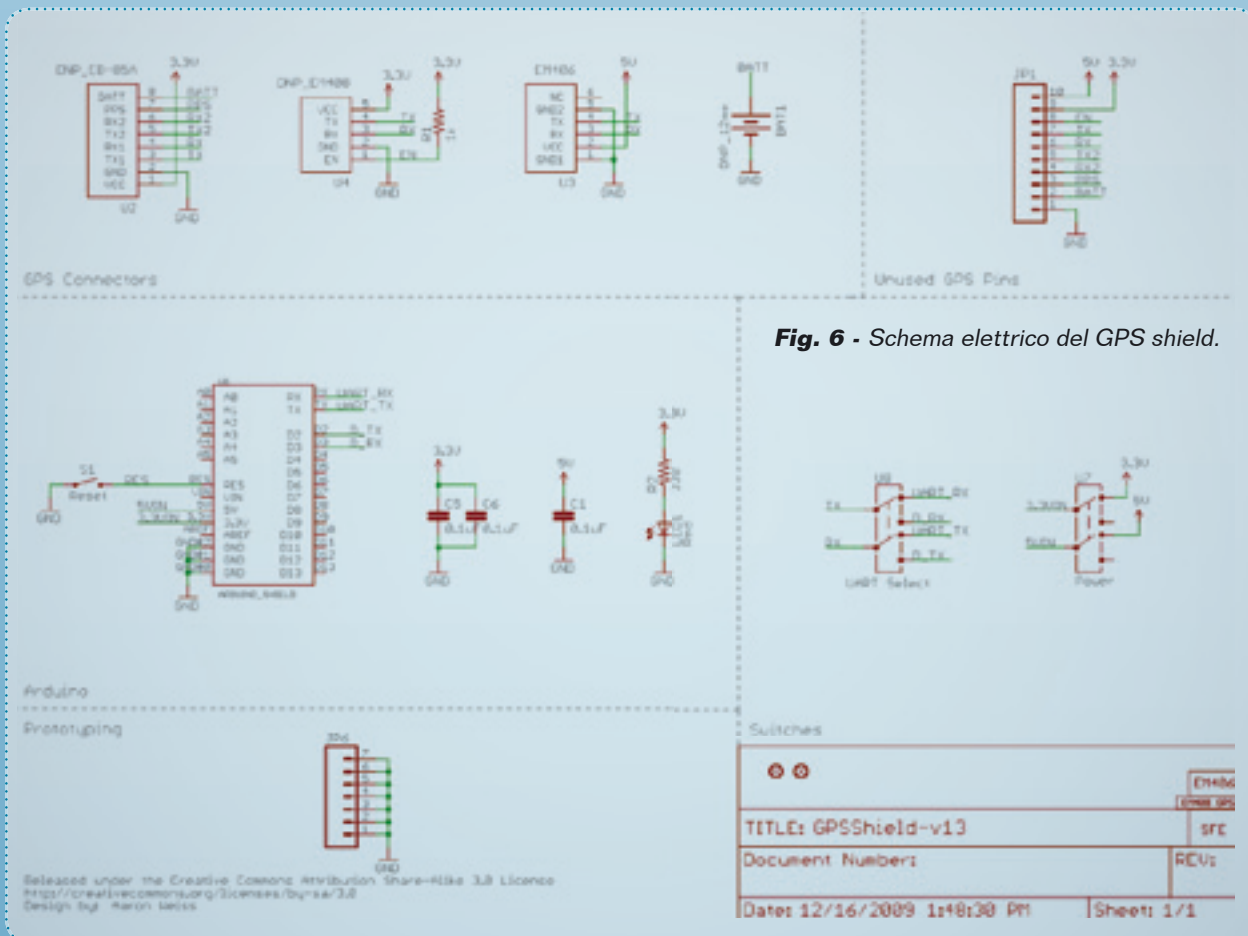
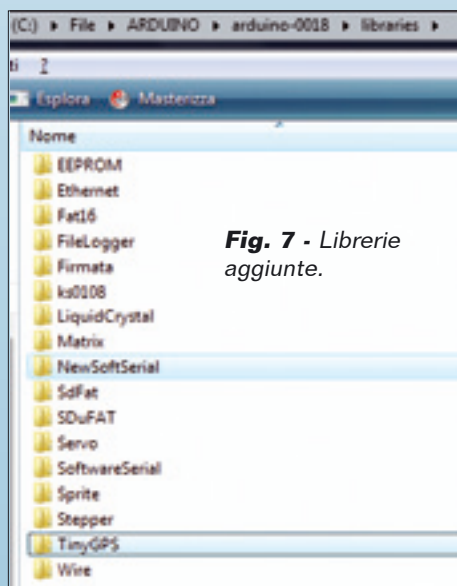
***Fig. 5** - GPS shield e relativi connettori.*

DLINE il ricevitore GPS è connesso alle linee digitali D2 e D3 di Arduino. Ponete il deviatore nella posizione DLINE sia per programmare Arduino sia per testare gli sketch; in questo modo non ci sarà alcun conflitto con i dati provenienti dalla USB, che invece impegnano le linee TX ed RX (pin 0 e 1).
Assicuratevi che il modulo EM406 non tocchi le piazzole sottostanti lo stampato della GPS-shield e in caso contrario isolatelo con della plastica.
A questo punto potete occuparvi della programmazione, obiettivo della quale è leggere la stringa che esce ogni secondo dal modulo EM406, estrapolando le varie sentenze contenenti i dati. Aver utilizzato la piattaforma

***Fig. 6** - Schema elettrico del GPS shield.*

Fig. 7 - Librerie aggiunte.

Fig. 8 - Schermata appena avviato l'esempio StaticTest.

Arduino ci assicura la disponibilità in rete di un'apposita libreria che ci assolva dal compito di dover redigere un complicato software.
Le librerie che andiamo ad utilizzare si chiamano NeoSoftSerial e TinyGPS, sono state create da Mikal Hart e risultano reperibili all'indirizzo web *http://arduiniana.org*; entrambe le librerie, come al solito, andranno copiate nella directory *libraries* del software Arduino. In alternativa potete scaricare queste librerie all'indirizzo *http://www.sparkfun.com/tutorial/GPSQuickStart/GPSQuickStart-lib.zip.*
Assicuratevi che il deviatore di alimentazione sulla GPS shield sia in posizione ON e che il secondo deviatore sia in posizione DLINK, poi collegate Arduino al PC con il solito cavo USB. Sulla GPS shield si deve accendere il LED rosso (alimentazione OK); il LED sul modulo EM406 darà le seguenti indicazioni:

- LED OFF = ricevitore spento;
- LED ON = non agganciato, ricerca del segnale;
- LED lampeggiante = segnale agganciato.

Aprite l'ambiente Arduino, importate l'esempio TinyGPS->StaticTest e caricatelo sul microcontrollore di Arduino. Aprite Tools->Serial Monitor ed impostate la comunicazione su 115.200 Baud; riceverete una serie di dati preimpostati dal software, ma non ancora provenienti dal modulo GPS. Questa prima fase vi assicura che il trasferimento e la visualizzazione dei dati siano corretti.
Scaricate quindi l'esempio denominato *gps_parsing_v12ii.pde* scritto da Aaron Weiss e reperibile all'indirizzo *http://www.sparkfun.com/tutorial/GPSQuickStart/gps_parsing_v12ii.pde*, poi caricatelo su Arduino; avviate Tool->SerialMonitor e finalmente a video vi ritroverete i dati acquisiti dal vostro GPS.
Se il LED del modulo EM406A rimane a luce fissa per più di un minuto, significa che siete

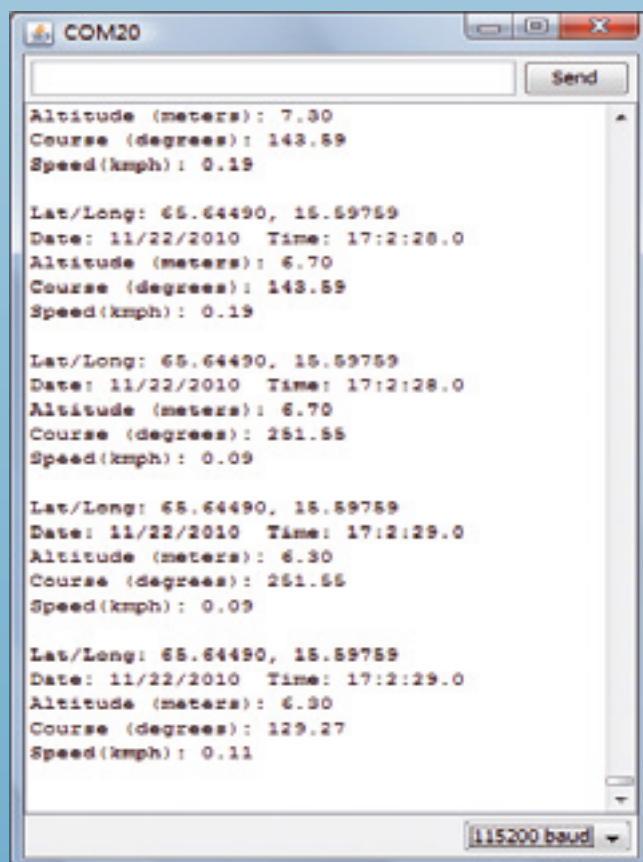



Fig. 9 - Schermata appena avviato lo sketch gps_parsing_v12ii.

Listato 1

```
// Queste righe servono ad includere le due librerie
        #include <NewSoftSerial.h>
        #include <TinyGPS.h>

// Si definiscono i pin usati per la comunicazione con Arduino ed il baud rate del modulo GPS
        #define RXPIN 3
        #define TXPIN 2
        #define GPSBAUD 4800

// Viene creta un'istanza all'oggetto TinyGPS
        TinyGPS gps;

// Si inizializza la libreria NewSoftSerial utilizzata per la comunicazione con il modulo GPS
        NewSoftSerial uart_gps(RXPIN, TXPIN);

// Si dichiara un prototipo per la funzione della libreria TinyGPS
        void getgps(TinyGPS &gps);

void setup()
{
        // Si inizializza il modulo UART per la comunicazione con il PC
                Serial.begin(115200);

        // Si imposta il baud rate per il modulo GPS
                uart_gps.begin(GPSBAUD);

        // Testo iniziale inviato dallo sketch verso il PC
        Serial.println("");
        Serial.println("GPS Shield QuickStart Example Sketch v12");
        Serial.println("       ...waiting for lock...           ");
        Serial.println("");
}

// Il loop principale semplicemente aspetta l'arrivo di una sentenza valida dal modulo GPS
// quindi ne estrae le sotto stringhe richieste e le invia al PC

void loop()
{
  while(uart_gps.available())     // Aspetta l'arrivo di dati validi
  {
      int c = uart_gps.read();       // Carica i dati ricevuti nella variabile c
      if(gps.encode(c))                // Verifica congruenza dati
      {
        getgps(gps);                     // Estrapola le sottostringhe di dati e le invia al PC
      }
  }
}

void getgps(TinyGPS &gps)
{

// Richiama la funzione che estrapola dalla sentenza i dati relativi alla posizione
        float latitude, longitude;
        gps.f_get_position(&latitude, &longitude);

// Invia al PC i dati estrapolati
        Serial.print("Lat/Long: ");
        Serial.print(latitude,5);
        Serial.print(", ");
        Serial.println(longitude,5);

// Richiama la funzione che estrapola dalla sentenza i dati relativi alla data
        int year;
        byte month, day, hour, minute, second, hundredths;
        gps.crack_datetime(&year,&month,&day,&hour,&minute,&second,&hundredths);

// Invia al PC i dati estrapolati
        Serial.print("Date: "); Serial.print(month, DEC); Serial.print("/");
        Serial.print(day, DEC); Serial.print("/"); Serial.print(year);
        Serial.print("  Time: "); Serial.print(hour, DEC); Serial.print(":");
        Serial.print(minute, DEC); Serial.print(":"); Serial.print(second, DEC);
        Serial.print("."); Serial.println(hundredths, DEC);

// Si può usare una sintassi diretta per inviare al PC i dati estratti
        Serial.print("Altitude (meters): "); Serial.println(gps.f_altitude());
        Serial.print("Course (degrees): "); Serial.println(gps.f_course());
        Serial.print("Speed(kmph): "); Serial.println(gps.f_speed_kmph());
        Serial.println();

}
```

in una posizione nella quale il segnale GPS non giunge con la sufficiente intensità. Spostatevi all'esterno, in una zona facilmente raggiungibile dai segnali satellitari, ed attendete che il modulo agganci correttamente i satelliti e cominci la fase di tracking nella quale, ogni secondo, invia i dati relativi alla posizione.
Vediamo ora in dettaglio come funziona questo sketch, che viene meglio descritto dal **Listato 1**. Si parte con due righe di codice che servono a includere le librerie *NewSoftSerial.h* e *TinyGPS.h* e, a seguire, vengono definiti i pin da utilizzare per acquisire i dati e il baud-rate corrispondente. Poi vengono create le istanze e inizializzate le librerie, quindi si inizializza l'UART e lo si imposta a 115.200 baud; ora parte il loop principale nel quale lo sketch attende l'arrivo di stringhe di dati dal lettore GPS. Quando questi arrivano, le elabora per estrapolare latitudine e longitudine (oltre ad altri dati significativi) e ne invia le informazioni al PC.
L'elenco completo dei dati estraibili dalla sentenza GGA in arrivo dal ricevitore GPS sono i seguenti:
- get_position;
- get_datetime;
- altitude;
- speed;
- course;
- stats;
- f_get_position;
- crack_datetime;
- f_altitude;
- f_course;
- f_speed_knots;
- f_speed_mph;
- f_speed_mps;
- f_speed_kmph;
- library_version.

Un primo semplice utilizzo dei dati acquisiti si può fare inserendo direttamente su Google Maps® le coordinate relative alla posizione, in modo da rintracciare il ricevitore sulle mappe di Google. Dalla riga in cui sono specificate la latitudine e la longitudine (Lat/Long:) estrapolate i due numeri successivi ed inseriteli nel seguente indirizzo internet: *http://maps.google.com;* per l'esattezza, componete nella casella degli indirizzi del browser una stringa del tipo *http://maps.google.com/maps?q=45.643876, 8.814163*, dove al posto di 45.643876, 8.814163 dovete scrivere le due coordinate desunte dalla vostra applicazione. Nel caso di questo esempio, 45.643876, 8.814163 sono le coordinate della nostra redazione.
Inserendo la stringa e premendo Invio (o facendo clic sul pulsante di inoltro dell'indirizzo) si aprirà la pagina web di Google Maps® con la posizione specificata sulla cartina sia in modalità stradale che satellitare, a scelta.
Per sperimentare applicazioni GPS con Arduino, potete usare un secondo modulo GPS denominato GPS Bee, facente uso del chipset U-BLOX5 e capace di ricevere sino a 50 satelliti contemporaneamente (ha quindi 50 canali); il suo formato lo rende del tutto compatibile con l'hardware dei moduli XBee. Le caratteristiche di questo modulo sono le seguenti:
- chipset U-BLOX 5;
- hot Start in 1 secondo;
- sensibilità di -160 dBm in acquisizione e tracking;
- precisione nella posizione di ±2 metri;
- precisione nel calcolo della velocità di ±0,1 m/s;
- avvio accelerato a segnali deboli per moduli con funzione Kickstart;
- supporto servizi AssistNow Online e AssistNow Offline A-GPS; compatibile OMA SUPL;
- elevata immunità ai disturbi;
- frequenza di aggiornamento della posizione = 4 Hz;
- interfacce UART, USB, DDC e SPI;
- alimentazione: 2,7 V ~ 3,6 V;

Fig. 10 - GPS Bee visto da sopra.

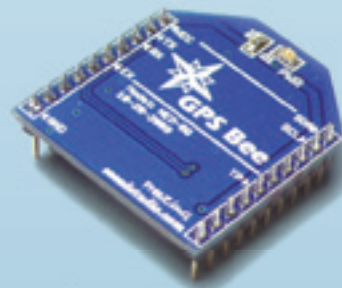



Fig. 11 GPS Bee visto da sotto.

• assorbimento: 44 mA;
• dimensioni XBee compatibili.

Questo modulo necessita di antenna esterna, come ad esempio la MINI ANTENNA GPS 15x15 mm con CONNETTORE U.FL, disponibile presso la ditta Futura Elettronica (codice Cod. 8160-MINIANTGPS) le cui caratteristiche sono:
• alimentazione: 3,3 V ~ 5 mA;
• V.S.W.R. (Rapporto Onde Stazionarie): <2;
• guadagno: 20 dB;
• peso: 15 g;
• dimensioni: 15 x 15 x 6,4 mm;
• lunghezza cavo: 90 mm.

Questo modulo GPS-Bee è facilmente interfacciabile con il PC utilizzando l'apposito adattatore XBee-USB (codice 7300-UARTSBV31 della Futura Elettronica). Grazie al software U-Center, scaricabile gratuitamente all'indirizzo *http://www.seeedstudio.com/depot/datasheet/u-center_5.07_Installer.rar*, è possibile visualizzare a video tutti i dati relativi a tempo, velocità, latitudine e longitudine, ecc.
Lo shield ArduinoXBee consente di interfacciare velocemente il modulo alla scheda Arduino, realizzando in maniera fissa le connessioni delle linee TX ed RX del modulo, alle linee TX ed RX della scheda Arduino, le quali, sono anche utilizzate dal convertitore USB-Seriale per l'interfacciamento alla porta USB. Questo vincolo non consente di gestire in contemporanea la comunicazione da parte del microcontrollore sia con il modulo GPS che con la porta USB; per i particolari, si veda la sesta puntata del corso, relativa ai moduli XBee.
Per la lettura dei dati in arrivo dal modulo GPS è possibile usare la libreria standard "Seriale" impostata per un Baud-Rate di 9.600 Baud, mentre la decodifica della sentenza può avvenire con la libreria precedentemente descritta.

Conoscere e usare Arduino

dell'ing. MIRCO SEGATELLO

Conosciamo e impariamo ad utilizzare Ethernet Shield, uno strumento che permetterà di affacciare Arduino su una rete o sul mondo di Internet. Decima puntata.

Proseguiamo spediti con il nostro corso sul mondo Arduino e questa volta ci occupiamo di sviluppare degli sketch orientati al web, tramite l'utilizzo dello Shield Ethernet. Abbinare i termini Arduino e web significa poter accedere ad un mondo di applicazioni davvero sorprendenti e di interessantissima utilità. In questo articolo vi dimostreremo come l'elettronica e l'informatica possano "andare a braccetto" e come l'una completi l'altra permettendo la realizzazione di sistemi, anche complessi, con una serie di applicazioni che spaziano dai controlli alla domotica all'informazione.

LO SHIELD ETHERNET

Chi conosce Arduino sa che oramai le applicazioni implementabili grazie ad esso non si contano più e che tra queste non avrebbe potuto mancare l'interfacciamento con una rete informatica. Se disponete di una scheda Arduino Diecimila, Duemilanove oppure Uno, avrete la possibilità di collegarvi ad una rete ethernet semplicemente applicando uno "Shield Ethernet". Ufficialmente Arduino ha adottato quale chip di interfaccia il modello W5100 prodotto dalla WizNet e quindi ci sarà piena compatibilità con i vari shield che utilizzano questo chip; tuttavia in commercio sono

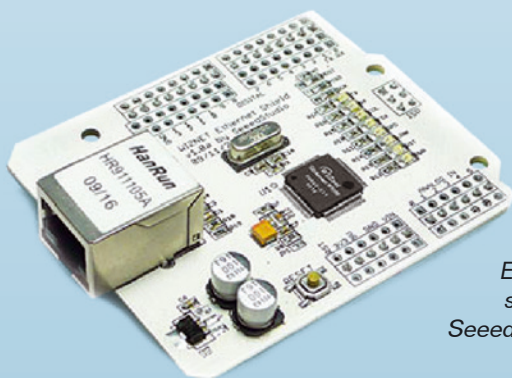



Fig 1 *Ethernet shield di SeeedStudio.*

disponibili altre interfacce equipaggiate con differenti chip. Lo Shield Ethernet che useremo in questa puntata è la versione prodotta da SeeedStudio, distribuita da Futura Elettronica (codice 7300-SEEEDWIZNET). Questa unità è basata sul chip Ethernet Wiznet W5100 ed è compatibile sia con Arduino (anche la versione MEGA) che Seeeduino (anche la versione MEGA11). Presso la stessa azienda è disponibile anche lo Shield Ethernet (codice 7300-SHIELDETHERNET) anch'esso basato sul chip Ethernet Wiznet W5100 e compatibile con Arduino (ma non con la versione MEGA). Entrambe gli shield fanno uso dello stesso chip e supportano fino a quattro connessioni socket simultanee; il connettore di collegamento è lo standard RJ45. I contatti utilizzati da Arduino per comunicare con il chip W5100 sono i pin digitali 10, 11, 12 e 13 (porta SPI) che quindi non possono essere utilizzati come I/O. Il pulsante di reset sulla scheda resetta sia il chip W5100 che la scheda Arduino (o Seeeduino). Il chip Ethernet Wiznet W5100 implementa a livello hardware uno stack IP completo di protocollo di comunicazione TCP ed UDP.
Il protocollo TCP (Transfer Control Protocol) si basa sulla creazione di una comunicazione tra un *Client* ed un *Server*. È il *Client* che può iniziare una comunicazione inviando una richiesta al *Server* che è in attesa su una porta in particolare; se il *Server* accetta, viene aperta una comunicazione e allora sarà possibile effettuare il trasferimento di dati.
Caratteristica di questo protocollo è quella di essere a conoscenza dello stato della connessione e di verificare se i pacchetti sono stati ricevuti correttamente dal ricevente.
Nel protocollo UDP (User Datagram Protocol), a differenza del TCP, non viene aperta una comunicazione, ma semplicemente è previsto l'invio di dati verso una determinata porta di un indirizzo IP; ciò significa che non esiste il ruolo di *Client* o *Server*.
Non vi è, quindi, alcun controllo dello stato della comunicazione (perché non è stata effettuata) né è possibile sapere se i dati inviati siano effettivamente giunti a destinazione.
Possiamo paragonare il protocollo UDP all'invio di una lettera tramite posta normale: non vi è certezza assoluta della consegna, ma nella stragrande maggioranza dei casi la trasmissione andrà a buon fine. Invece il protocollo TCP può essere visto come l'invio di una lettera tramite posta con ricevuta di ritorno, nel qual caso si ha una verifica della corretta ricezione del messaggio. Risulta più semplice, quindi, implementare il protocollo UDP, sebbene esso accentui i limiti del sistema.
Detto ciò, ritorniamo a parlare del nostro shield, elencando le funzioni dei LED posti a bordo:

- TX:lampeggia quando lo shield trasmette dati;
- RX: lampeggia quando lo shield riceve dati;
- COL: lampeggia quando si verifica una collisione di pacchetti all'interno della rete;
- FDX: indica che la connessione alla rete avviene in full-duplex;
- LINK: indica la presenza di una rete e lampeggia quando lo shield trasmette o riceve dati;
- RST: stato di reset;
- PWR: indica che lo shield è alimentato.

Per quanto riguarda l'utilizzo dello shield a livello software, non ci sono problemi in quanto l'ambiente di lavoro di Arduino contiene già la libreria e gli esempi per la gestione della Ethernet; possiamo quindi passare direttamente all'impiego pratico. Lo Shield Ethernet (versione SeeedStudio) viene fornito già pronto all'uso, ma consigliamo di saldare sugli appositi spazi degli strip femmina, in modo da avere facilmente accesso ai pin di Arduino o per inserirci un secondo shield; appena inserita nella scheda Arduino, è pronta all'uso. Durante tutte le prove che vorrete fare, potrete lasciare collegato sia il cavo di rete che quello USB

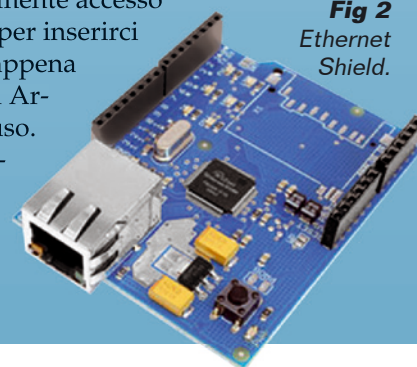



Fig 2 *Ethernet Shield.*

usato per la programmazione; infatti Arduino dialoga con lo Shield Ethernet tramite la porta SPI, lasciando libero l'UART.
Particolare attenzione deve essere prestata alla configurazione della vostra rete domestica; giusto per esservi di aiuto, vi diciamo che sul PC principale da noi usato per programmare Arduino è installato il sistema operativo Vista e che i parametri della rete sono IP=192.168.0.199 e SubNetMask=255.255.255.0, mentre Gateway e Server DNS non sono specificati. Queste impostazioni sono gestibili accedendo a: Start>Impostazioni>Pannello di controllo>Centro di connessione di rete e condivisione>Gestisci connessioni di rete.
Il PC e Arduino (Ethernet Shield) sono connessi ad un router D-Link con impostazioni da fabbrica. Come IDE di sviluppo facciamo riferimento alla versione 0021; se usate versioni più vecchie, alcune funzioni, specificamente sulle stringhe, non saranno implementabili.
Come primo semplice esempio utilizzeremo il nostro Ethernet Shield per realizzare un Web Server.
Detto in parole semplici, andremo a creare tramite il chip W5100 una pagina web accessibile in rete da qualsiasi browser, che conterrà i dati relativi al valore degli ingressi analogici di Arduino; quindi non una semplice pagina web, ma una speciale pagina che sia in grado di interagire con Arduino e quindi con il mondo esterno.
Collegate il tutto, avviate l'IDE di Arduino ed aprite l'esempio *File-Esempio-Ethernet-WebServer*; per rendere compatibile la pagina web con la nostra rete, dovremo impostare correttamente il MAC e l'IP address. L'indirizzo MAC (MAC address) è un codice di 12 caratteri che identifica ogni singolo apparato in grado di connettersi a una rete; nel nostro caso, sia la scheda di rete che il router ne avranno uno univoco. Lasciate invariato quello proposto nello sketch. Solo se la vostra scheda dovesse accedere alla rete internet, dovrete preoccuparvi di recuperare un indirizzo MAC libero perché altri potrebbero aver avuto la vostra stessa idea.
L'indirizzo IP (IP Address) è una sequenza di quattro numeri compresi tra 0 e 255 e, similmente al MAC, identifica ogni dispositivo all'interno di una rete. Anche questo deve essere univoco per ogni dispositivo sulla rete e va assegnato in congiunzione con il *subnet mask*, il quale è un'altra sequenza di quattro numeri che permettono di specificare delle sottoreti. Nel nostro caso, avendo una *subnet mask* pari a 255.255.255.0 potranno colloquiare tra di loro tutti i dispositivi facenti parte della sottorete che ha indirizzo IP con i primi tre numeri uguali ed il terzo scelto a piacere e compreso tra 0 e 255. L'indirizzo IP di Arduino dovrà quindi essere del tipo 192.168.0.xxx, dove xxx sarà un numero compreso tra 0 e 255 non utilizzato da altri dispositivi nella rete.
In definitiva, la configurazione per il nostro esempio sarà:

ROUTER	IP=192.168.0.1
PC	IP=192.168.0.199
PC	subnet mask=255.255.255.0
ARDUINO	IP=192.168.0.77
ARDUINO	subnet mask=255.255.255.0

Lasceremo non specificati tutti gli altri parametri.
In una connessione TCP, oltre all'indirizzo IP della postazione remota cui devono giungere le informazioni, bisogna conoscere anche a quale applicazione esse debbono essere passate. A tale scopo, si usano le porte, le quali sono una specie di allocazione della memoria del computer che esiste quando due computer sono in comunicazione tra loro e forniscono un punto terminale per l'applicazione remota. Le porte vengono sempre identificate con un numero compreso tra 0 e 65.535, i cui valori 0

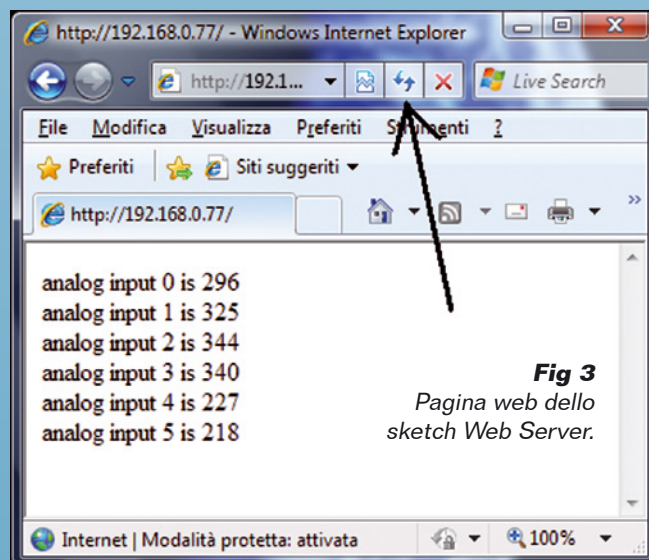



Fig 3
Pagina web dello sketch Web Server.

e 1.023 sono già assegnati a specifiche funzioni del sistema operativo.
Ecco la configurazione usata per il nostro esempio:

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
byte ip[] = { 192, 168, 0, 77 };
Server server(80);
```

Ora caricate lo sketch su Arduino e lasciate che il nuovo firmware avvii la comunicazione con il router. Aprite Internet Explorer o un Browser a scelta e scrivete l'indirizzo *http://192.168.0.77*: vedrete caricarsi la pagina web residente all'interno di Arduino, riportante i dati relativi agli ingressi analogici.
Se agli ingressi non avete collegato alcunché, i valori saranno casuali ed in ogni caso per leggere una nuova serie di valori sarà necessario rinnovare la richiesta di lettura pagina, cosa fattibile cliccando sul pulsante di *refresh*. Per testare meglio lo sketch, utilizzate un trimmer del valore di 10 kohm, al quale avrete saldato tre fili che collegherete così: gli estremi ai pin +5V, GND ed il centrale ad uno degli ingressi analogici. Utilizzate il trimmer per impostare un certo livello di tensione in ingresso.

L'applicazione è così interessante che viene voglia di migliorarla, completando la pagina web ed attivando un autorefresh in modo da visualizzare in tempo reale il livello di tensione all'ingresso. Poche conoscenze di HTML permettono di scrivere il nuovo sketch, il quale si chiama *Ethernet_01.pde*, che potete anche scaricare dal nostro sito *www.elettronicain.it* assieme agli altri file di questo corso; per facilitarne la comprensione i commenti sono stati tradotti in italiano.
A questo punto vogliamo imparare a gestire le uscite e lo facciamo inserendo dei pulsanti sulla pagina web, con i quali poter attivare un semplice LED. Essendo occupata l'uscita 13, il LED di sistema non sarà accessibile, perciò per le segnalazioni del caso usate un semplice LED con in serie una resistenza da 470 ohm, connesso tra l'uscita digitale 2 ed il pin di massa. Lo sketch già commentato in italiano si chiama *ethernet_02.pde*.
In questo sketch è stata aggiunta la possibilità di gestire il LED sia tramite un *check*, sia mediante due pulsanti; la pagina web è anche più complessa, proprio per far vedere le potenzialità del sistema. Volendo approfondire l'argomento, esiste un'interessante libreria

http://192.168.0.77/

ElettronicaIN

analog input 0 is 330
analog input 1 is 309
analog input 2 is 310
analog input 3 is 317
analog input 4 is 471
analog input 5 is 446

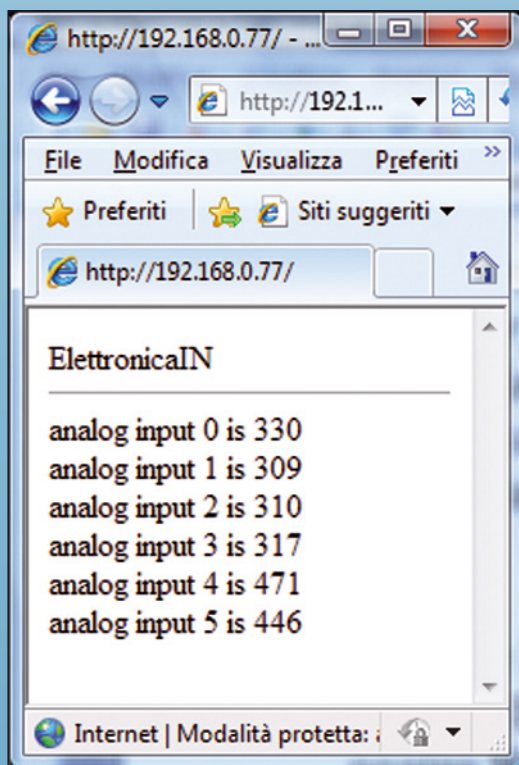



Fig 4
Pagina web dello sketch Ethernet_01.pde.

Fig 5
Pagina web dello sketch Ethernet_02.pde.

creata da Jordan Terrell (*http://blog.jordanterrell.com*) la quale permette di implementare la funzione di DHCP, ovvero la possibilità che il router assegni alla scheda Arduino in automatico un indirizzo IP libero senza doverlo specificare a priori.
Se volessimo rendere visibile la periferica da noi creata anche al di fuori della nostra rete domestica, dovremmo necessariamente disporre di un collegamento alla rete internet ed abilitare un canale di comunicazione con la nostra applicazione; infatti gli accessi dall'esterno saranno intercettati dal router, il quale dovrà essere configurato affinché un certo tipo di richieste giunga proprio ad Arduino. Per fare questo si deve attivare il servizio di "redirezione port forwarding" del router, impostando il numero di porta accessibile da internet e specificando a quale porta effettivamente è connesso Arduino sulla rete interna. Per i dettagli, vi rimandiamo alle specifiche del tipo di router che utilizzate.
Sempre accedendo alla rete internet, potete testare l'esempio WebClient, il quale prevede che sia la nostra applicazione a fare una richiesta (*Client*) verso un *Server* esterno: Google, in questo caso. La richiesta consiste nella ricerca del termine Arduino, mentre la risposta visualizzata tramite SerialMonitor sarà il risultato della ricerca.
Un altro interessantissimo esempio si chiama UdpNtpClient e prevede, sempre tramite accesso ad internet, di interrogare un *Server Ntp* per rilevare l'ora internazionale sempre precisa. Infatti i server Ntp (Network Time Protocol) dispongono di informazioni orarie messe a disposizione degli utenti e facilmente accessibili tramite l'invio di richieste in protocollo UDP.
L'ora corrente viene letta dall'applicazione e successivamente resa disponibile tramite SerialMonitor di Arduino.
Esiste un'altra possibilità per rendere disponibili i dati acquisiti da Arduino verso Internet, ovvero trasferire i dati ad un server di dati, come ad esempio il server Pachube, accessibile dall'indirizzo web *www.pachube.com;* in questo caso Arduino dialogherà solo con detto server, mentre gli utenti che vorranno vedere i dati accederanno al sito *www.pachube.com*, il quale, oltre a rendere disponibili i dati, fornirà una visualizzazione in formato grafico

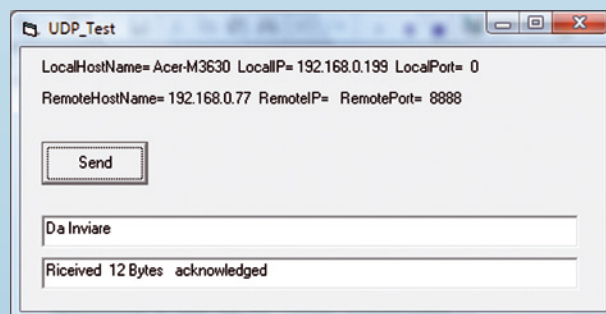



Fig 6
Applicazione del protocollo UDP in Visual Basic.

ed informazioni di geolocalizzazione.
Per quanto riguarda lo sviluppo del firmware, Arduino mette a disposizione gli esempi *PachubeClinet* e *PachubeClientString*. Un valido esempio è già stato proposto sul numero 151 della rivista a proposito dell'articolo riguardante il "Termostato ambiente".
È disponibile anche un altro interessantissimo esempio che prevede l'invio e la ricezione di stringhe dal PC ad Arduino con il protocollo UDP: aprite lo sketch denominato *UDPSendReceive.pde*, modificate se necessario l'indirizzo IP e caricatelo su Arduino.
Nella parte terminale dello sketch trovate il listato da far girare su processing (lo trovate nella quinta puntata di questo corso su Arduino), utilizzato in questo caso per inviare la stringa "hello word" via ethernet alla nostra scheda Arduino. Tramite Serial monitor dell'IDE di Arduino vedrete la stringa arrivata ed una risposta di consenso giungerà anche a processing. Per poter usare il protocollo UDP con processing è necessario installare la libreria *hypermedia.net*, che trovate nel nostro sito assieme ai file di questo numero della rivista.
Vista l'interessante modalità di comunicazione tra un PC ed Arduino tramite ethernet abbiamo pensato di realizzare una piccola applicazione scritta in Visual Basic 6 che implementa, appunto, il protocollo UDP. Anche in questo caso l'applicazione invia una semplice stringa di testo ed attende risposta da Arduino, ma facilmente può essere modificata per ogni altra esigenza.
Come avete visto, le applicazioni implementabili sono davvero moltissime e tutte della massima facilità. Alcune conoscenze sulle reti informatiche e di programmazione in HTML permettono di ottenere in brevissimo tempo delle applicazioni molto innovative e professionali che non mancheranno di darvi soddisfazione.

Conoscere e usare Arduino

dell'ing.
MIRCO
SEGATELLO

Conosciamo ed impariamo ad usare un nuovo shield: quello che permette di dotare il nostro Arduino della connettività Bluetooth. Undicesima puntata.

Ogni giorno o quasi, nascono shield dedicati a realizzare con Arduino tutte le funzioni che già conosciamo e vediamo associate a dispositivi di uso comune. In questa puntata impareremo ad utilizzare quello creato e sviluppato per implementare il Bluetooth. Per la precisione, vogliamo interfacciare una scheda Arduino ad un PC tramite un collegamento wireless utilizzando lo standard Bluetooth, già impiegato con successo nella comunicazione locale tra cellulari, negli auricolari e vivavoce wireless per automobile, nella strumentazione, ma anche nelle periferiche del computer e nei lettori di codice a barre senza fili. Come vedremo, dotare Arduino di un'interfaccia Bluetooth risulterà assai semplice e per niente costoso; sarà inoltre una valida soluzione per comunicazione wireless, alternativa a quella rappresentata dai moduli radio XBee o WiFi. Come sempre, in questo corso, partiamo dalla descrizione dell'hardware, per poi proseguire con lo sviluppo della parte software.

In sintesi, gli elementi che abbiamo utilizzato per questa puntata, sono i seguenti:

- una scheda Arduino duemilanove (va bene anche Arduino UNO);
- una XBee shield, che è la versione prodotta dalla Libelium (*www.libelium.com*);
- un modulo Bluetooth Bee della Seeedstudio

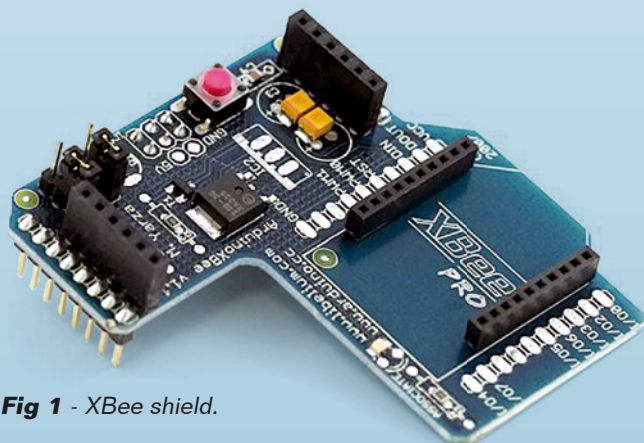



Fig 1 - XBee shield.

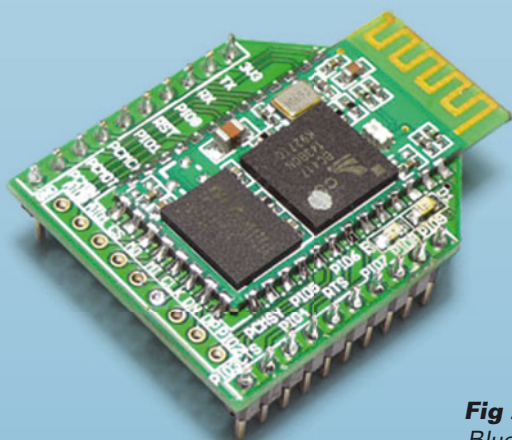



Fig 2 - Modulo Bluetooth Bee.

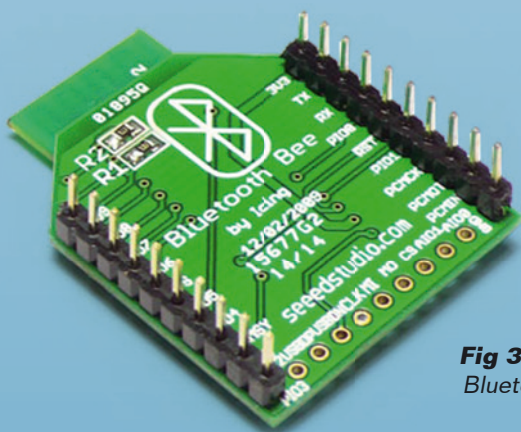



Fig 3 - Modulo Bluetooth Bee.

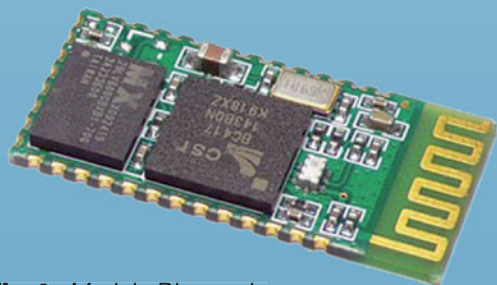



Fig 4 - Modulo Bluetooth.

(*www.seeedstudio.com*) fornito dalla Futura Elettronica (codice 7300-BLUETBEE).

È possibile acquistare a parte il solo modulo Bluetooth (Futura Elettronica, codice 7300-BLUETOOTHMOD) ma, viste le ridotte dimensioni, il suo utilizzo non è per niente agevole. Pur costando di più, consigliamo l'utilizzo del modulo già cablato sulla piccola basetta XBee, facilmente utilizzabile con la sua XBee shield.
Vediamo ora brevemente le caratteristiche di questo ricetrasmettitore Bluetooth appositamente progettato per realizzare collegamenti trasparenti tra due apparecchiature. Questo modulo contiene un ricetrasmettitore funzionante nella banda ISM (ricordiamo che lo standard Bluetooth riguarda connessioni effettuate alla frequenza di 2,4 GHz) ed è in grado di gestire collegamenti seriali sino a 460.800 bps.
È compatibile con le specifiche Bluetooth v2.0 + EDR, con una potenza in trasmissione di 4 dBm (Classe 2) che gli consente collegamenti sino a circa 10 metri.
Il modulo UART interno ha il baud-rate programmabile e la gestione del transito dati è asservita ad un sistema di crittografia; l'alimentazione è a 3,3 volt e dispone di un'antenna integrata nel PCB.
Il modulo UART interno è configurato, per impostazione predefinita, per una comunicazione a 38.400 baud con 8 bit di dati, 1 bit di stop, nessuna parità, nessun controllo di flusso; sono comunque supportate le seguenti velocità di trasferimento: 9.600, 19.200, 38.400, 57.600, 115.200, 230.400, 460.800.
Sono disponibili le linee CTS e RTS per il controllo del flusso dati e due LED (uno verde e uno rosso) per la segnalazione dello stato di funzionamento.
Quando il modulo si trova nello stato "sconnesso", si vedrà il solo LED verde lampeggiare 2 volte al secondo. In attesa di collegamento ci saranno sia il LED verde che il LED rosso a lampeggiare una volta la secondo, mentre nello stato "connesso" lampeggerà il solo LED verde una volta al secondo.
Per impostazione predefinita, il modulo è predisposto alla connessione automatica con l'ultimo dispositivo non appena viene acceso; il codice di accesso è impostato a "0000". Per

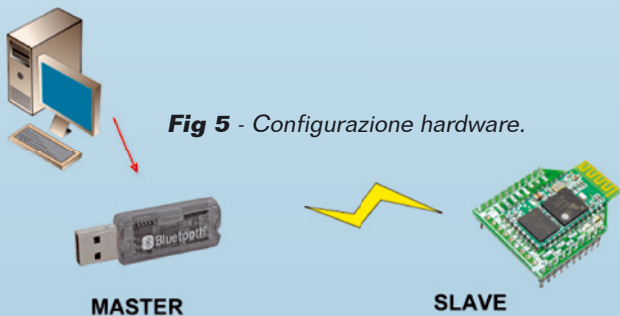



Fig 5 - *Configurazione hardware.*

"ultimo dispositivo" si intende quello con cui si è collegato prima di essere spento.
Per i dettagli relativi alla funzione dei pin di collegamento, vi rimandiamo alla lettura dei data-sheet del prodotto.
Volendo realizzare un collegamento wireless con un PC, dovremmo necessariamente dotarci di un adattatore Bluetooth (Bluetooth dongle) da inserire in una porta USB libera del PC. Per le nostre verifiche sul campo abbiamo utilizzato il Bluetooth USB dongle prodotto dalla Velleman e distribuito dalla Futura Elettronica con il codice PCUSBBT; esso viene fornito con il software Bluesoleil della IVT Corporation (il più diffuso per far dialogare dispositivi Bluetooth con un computer) al quale faremo riferimento in questa puntata del corso. Il modulo Bluetooth Bee sarà inserito nella XBee shield e a sua volta su Arduino, ma, per poterlo utilizzare, è necessario programmarlo; la programmazione avviene tramite l'invio di semplici comandi seriali. È possibile inserire il modulo Bluetooth Bee in un adattatore USB-XBee, in modo da consentire la programmazione via PC, ma preferiamo sia direttamente Arduino, tramite un apposito sketch, ad inviare le impostazioni. Elenchiamo qui di seguito i comandi a disposizione.

Imposta modalità di funzionamento
\r\n+STWMOD=0\r\n   Client (slave)
\r\n+STWMOD=1\r\n   Server (master)

Imposta Baud-rate
\r\n+STBD=115200\r\n   Imposta baudrate 115200
Baudrate supportati: 9600, 19200,38400,57600,115200,230400,460800.

Imposta il nome del dispositivo
\r\n+STNA=abcdefg   Imposta il nome "abcdefg"

Listato 1

```
/*
BluetoothBee Demo Code
2011 ElettronicaIN

Hardware:
  modulo BluetoothBee
  XBee shiled Libelium
  Arduino duemilanove
  USB Bluetooth dongle Velleman

Questo sketch configura il modulo come slave
*/


void setup()
{
  Serial.begin(38400); //Imposta l'UART a 38400 baud
    delay(1000);
    Serial.print("\r\n+STWMOD=0\r\n");  //Imposta il modulo come slave
    Serial.print("\r\n+STNA=Arduino\r\n"); //Assegna il nome "Arduino" al modulo
    delay(2000); // Ritardo necessario per la configurazione.
    Serial.print("\r\n+INQ=1\r\n");
    delay(2000); // Ritardo necessario per abilitare la comunicazione.
}

void loop()
{
  delay(1000);
  int sensorValue = analogRead(A0);  //Legge il canale analogico A0
  Serial.println(sensorValue, DEC);  //Invia il dato via UART
}
```

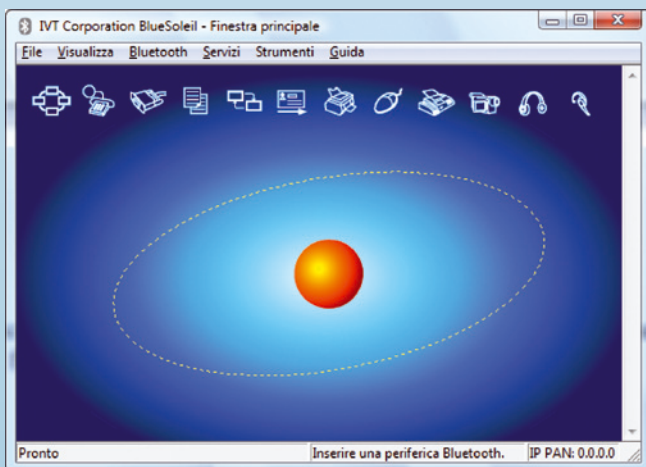



Fig 6 - *Schermata di avvio di Bluesoleil.*

Fig 7 - *Ricerca periferiche.*

Autoconnessione con l'ultimo dispositivo connesso
\r\n+STAUTO=0\r\n Proibito
\r\n+STAUTO=1\r\n Permesso

Permette la connessione del dispositivo
\r\n+STOAUT=0\r\n Proibito
\r\n+STOAUT=1\r\n Permesso

Imposta PIN code (Passkey)
\r\n +STPIN=2222\r\n Imposta il PINCODE "2222"

Cancella PIN code (Passkey assegnata dal microcontrollore)
\r\n+DLPIN\r\n Cancella PINCODE

Legge l'indirizzo del dispositivo
\r\n+RTADDR\r\n Riporta l'indirizzo del dispositivo

Fig 8 - *Ricerca servizi della periferica trovata.*

Fig 9 - *Inserimento Passkey.*

Auto riconnessione quando viene perso il segnale con il dispositivo master
\r\n+LOSSRECONN=0\r\n Proibito
\r\n+LOSSRECONN=1\r\n Permesso

Ricerca moduli
a) Nel caso di modulo configurato come Master:
\r\n+INQ=0\r\n Ferma ricerca dispositivi
\r\n+INQ=1\r\n Avvia ricerca dispositivi

b) Nel caso di modulo configurato come Slave:
\r\n+INQ=0\r\n Ferma visibilità dispositivo
\r\n+INQ=1\r\n Avvia visibilità dispositivo

Siamo adesso pronti per scrivere lo sketch occorrente alla gestione della comunicazione in Bluetooth, il quale, una volta caricato su Arduino, configurerà il modulo e, successivamente, andrà a leggere l'ingresso analogico 0 ed invierà al PC i dati acquisiti, ad intervalli di un secondo, via Bluetooth. Il codice corrispondente è quello illustrato nel **Listato 1**.
La configurazione del modulo avviene inviando semplici caratteri seriali nello standard

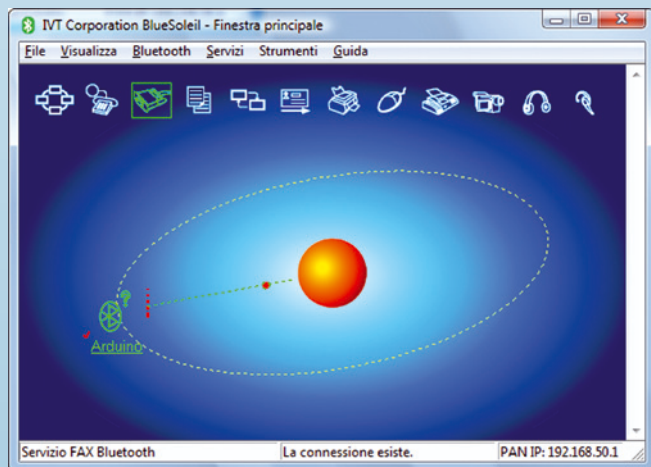



Fig 10 *- Connessione stabilita.*

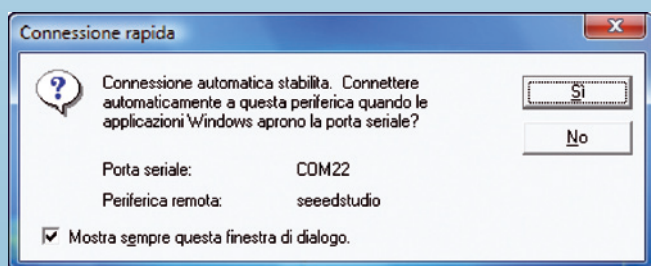



Fig 11 *- Porta COM virtuale utilizzata dalla connessione.*

Fig 12 *- Dati ricevuti da Serial monitor di Arduino.*

specificato nei data-sheet. Il comando "*\r\n+STWMOD=0\r\n*" imposta il modulo come Slave; sarà l'adattatore Bluetooth sul PC a svolgere la funzione di Master.
Il comando "*\r\n+STNA=Arduino\r\n*" assegna al modulo il nome "Arduino"; infine, il comando "*\r\n+INQ=1\r\n*" abilita la visibilità del modulo da parte del Master.
Tutte queste fasi sono chiaramente indicate dallo stato di lampeggio dei LED. Quando il LED verde lampeggia due volte al secondo, significa che il modulo non è connesso; ciò accade, ad esempio, quando alimentate quest'ultimo senza prima averlo configurato.
Quando si abilita la visibilità del modulo, lampeggiano una volta al secondo i LED sia verde che rosso, mentre una volta che viene stabilita una connessione, lampeggia il solo LED verde , sempre con la cadenza di una volta al secondo. Durante la fase di programmazione di Arduino con lo sketch riportato nel **Listato 1** è importante che gli interruttori della XBee shield siano posti in modalità USB. Quindi staccate Arduino, impostate gli interruttori della XBee Shield su XBee ed alimentate la scheda tramite la presa Plug. Questo è necessario perché il modulo Bluetooth Bee dialoga con il microcontrollore di Arduino tramite il modulo UART, il quale non deve entrare in conflitto con eventuali segnali giunti via USB.
Passati alcuni secondi dall'accensione, il modulo Bluetooth sarà operativo e visibile. Avviate il software Bluesoleil, installato precedentemente, inserite il Bluetooth dongle, attendete venga riconosciuto, quindi selezionate Bluetooth-Rileva periferiche Bluetooth; sul modulo Bluetooth Bee devono lampeggiare entrambi i LED. Cliccate con il pulsante destro sopra l'icona della periferica trovata e, dal menu contestuale cui accedete, attivate la funzione di ricerca servizi. Verrà chiesta la Passkey di accesso alla periferica; il valore predefinito è 0000 (quattro zeri). Sempre con il pulsante destro del mouse sopra la periferica, dal menu contestuale avviate il servizio SPP (servizio porta seriale). A questo punto inizierà a lampeggiare lentamente il solo LED verde. Visualizzate le proprietà della connessione SPP per sapere a quale COM virtuale è stata associata la comunicazione con il modulo.
Avviate l'IDE di Arduino, aprite Serial monitor sulla COM, assegnata all'adattatore Bluetooth, con il Baud Rate di 38.400 ed attendete l'arrivo dei dati acquisiti da Arduino. Notate che è come se Arduino fosse collegato via cavo USB, ma in pratica è connesso in modalità wireless. Non essendo cablate le linee RTS e CTS dal modulo Bluetooth Bee verso Arduino, non sarà possibile la programmazione di Arduino tramite questo collegamento.

Conoscere e usare Arduino

dell'ing.
MIRCO
SEGATELLO

Sperimentiamo la realizzazione di una connessione Wi-Fi, resa possibile dallo specifico modulo WiFly Shield, prodotto dalla Rovin Network e compatibile anche con alcuni cloni di Arduino. Dodicesima e ultima puntata.

In questa puntata del corso su Arduino ci occupiamo dello shield denominato WiFly, che permette ad una scheda Arduino (anche alle versioni "clone" con essa compatibili) di potersi connettere ad una rete wireless secondo lo standard 802.11b/g. Lo shield che abbiamo utilizzato per questa occasione è stato sviluppato dalla ditta americana Sparkfun ed è basato sul modulo wireless prodotto dalla Roving Network di sigla RN-131G. Questo modulo, operante in logica a 3,3 volt, potrebbe esse interfacciato con il modulo seriale (UART) di Arduino, ma ciò non garantirebbe la possibilità di utilizzare la massima velocità di comunicazione. Per sfruttare appieno le prestazioni del modulo wireless è preferibile interfacciarlo con la porta SPI (Serial Peripheral Interface Bus) di Arduino, che assicura una velocità di trasferimento ben superiore rispetto a quella garanti-

ta dalla trasmissione asincrona dell'UART. Il convertitore da UART a SPI siglato SC16IS750, permette la comunicazione tra il modulo UART ad alta velocità dell'integrato RN-131G, con la porta SPI di Arduino, facente uso dei segnali CS, MOSI, MISO, SCLK che utilizzano i pin 10, 11, 12, 13.
L'interfaccia UART di Arduino rimane così a disposizione per la comunicazione con il PC.
La differente configurazione del convertitore SC16IS750 ha portato all'esistenza in commercio di tre differenti versioni dello stesso WiFly shield.
La versione più recente (Revision 3) è distribuita dalla ditta Sparkfun (*www.sparkfun.com/products/9954*), viene contraddistinta dal codice 9954 e riporta sul lato inferiore la data "6/15/10". In questa versione è presente un pulsante di reset ed il quarzo utilizzato per l' SC16IS750 è da 14 MHz. La versione meno recente (Revision 2) (*www.sparkfun.com/products/9367*) ha codice 9367 e si riconosce per la presenza del quarzo a 14 MHz e per un piccolo dispositivo metallico rettangolare vicino ai pin 6 e 7, riportante la scritta "14."; sullo stampato non è presente alcuna data stampigliata e non c'è il pulsante di reset. La primissima versione (Revision1) si riconosce per la presenza di un quarzo da 12 MHz e per la scritta "12." serigrafata vicino ai contatti 6 e 7. La corretta impostazione del tipo di shield utilizzato deve essere specificato con la variabile *SHIELD_REVISION* presente nel file *Configuration.h* della libreria (si veda la parte riguardante il software di gestione, in questa stessa puntata).
Vediamo ora le caratteristiche implementate nel modulo WiFly della Rovin Network: si tratta di un modulo a bassissimo consumo (non superiore ai 100 mW) studiato per funzionare con alimentazione a batteria e in grado di interagire in reti wireless secondo lo standard 802.11b/g. Utilizza la banda radio dei 2,4 GHz e dispone di uno stack TCP/IP, un real-time clock, un'interfaccia analogica e della cifratura dei dati. Nella configurazione hardware più semplice sono sufficienti quattro linee (+Vcc, TX, RX, GND) per creare una connessione wireless. La distanza di trasmissione, a seconda delle condizioni, arriva ad un massimo di 100 metri. Queste, in sintesi, le caratteristiche del modulo:

- bassissimo consumo: 4 µA sleep, 35 mA RX e 200 mA massimi in TX;
- velocità di trasferimento fino a 1 Mbps tramite UART;
- supporto per reti "ad hoc";
- interfaccia hardware UART ad alta velocità;
- 10 linee digitali per uso generico, programmabili;
- 8 linee analogiche per interfacciamento con sensori;
- real-time clock, auto-sleep e modalità auto-wakeup;
- alimentazione a 3,3 Vcc oppure 2÷3 Vcc con batterie;
- configurazione tramite UART o wireless con semplici comandi ASCII;
- aggiornamento firmware tramite FTP;
- autenticazione di rete WEP-128, WPA-PSK (TKIP), WPA2-PSK;

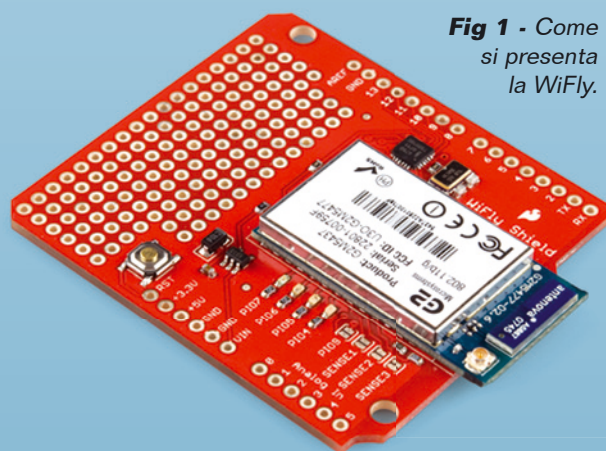



Fig 1 *- Come si presenta la WiFly.*

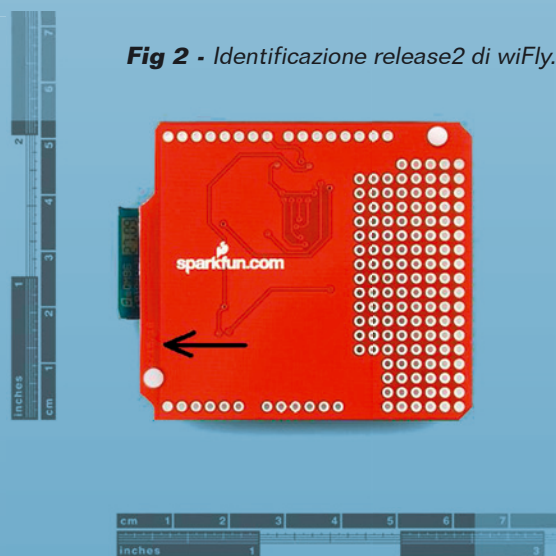



Fig 2 *- Identificazione release2 di wiFly.*

- protocolli di rete implementati: DHCP, UDP, DNS, ARP, ICMP.

Lo shield utilizzato per questo articolo è l'ultima versione, viene fornito già montato, e in esso è sufficiente saldare gli appositi strip per permettere la facile installazione su Arduino ed eventualmente sovrapporre un secondo shield. Il sito di riferimento per quanto riguarda la parte software è: *www.sparkfun.com/commerce/tutorial_info.php*. Da esso, oltre ai vari esempi, potete seguire il forum relativo all'argomento ed il link riguardante lo sviluppo del software di gestione.
Una prima vera libreria non esiste, in quanto è ancora in fase di sviluppo da parte della comunità di Arduino, però la versione "alpha2" è già a buon punto ed in fase di test da parte degli utilizzatori di Arduino, cui anche noi apparteniamo).
Non ci resta che inserire il WiFly shield su Arduino e connettere Arduino al PC, con il solito cavo USB. Scaricate la libreria dal sito *https://github.com/sparkfun/WiFly-Shield*; il file da noi utilizzato si chiama *sparkfun-WiFly-Shield-wifly-library-alpha-2-11-g981ea95.zip*. Scompattatelo (mantenendo inalterata la struttura delle sotto directory) e copiate la cartella *wifly* nella cartella *library* di Arduino. Aprendo L'IDE di Arduino vi ritroverete con una serie di esempi già preparati; aprite ed inviate ad Arduino quello denominato *SpiUartTerminal*, il quale permette la gestione a comandi del WiFly shield. Questo semplicissimo sketch vi permette di verificare il corretto interfacciamento, tramite l'integrato SC16IS750, tra Arduino e il modulo WiFly ed abilita la programmazione manuale del modulo RN-131G.
Aprite *Serial-Monitor* ed attendete la connessione al convertitore SPI-UART; se tutto è andato bene otterrete una risposta positiva; in caso contrario verificate la versione del vostro shield e modificate di conseguenza il file *configuration.h*. A questo punto inviate tre caratteri dollaro "$$$" in modo da portare il modulo WiFly in modalità di comando; esso deve rispondere con la stringa CMD.
A questo punto, in serial monitor, passate dalla modalità *no line ending* alla modalità *carriage return*, in quanto i prossimi comandi devono terminare con un *fine linea*. Inviate il comando "ver" ed otterrete come risposta la versione

Fig 3 *- Avvio sketch SpiUartTerminal.*

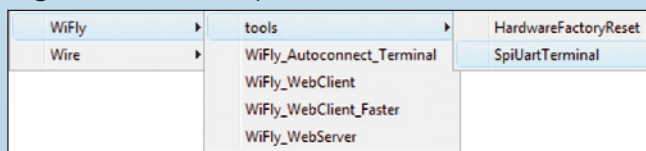



Fig 4 *- Modalità di accesso al WiFly con comandi.*

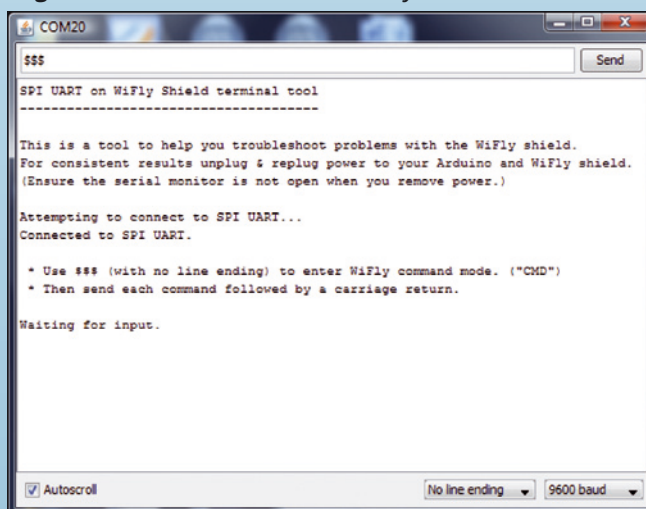



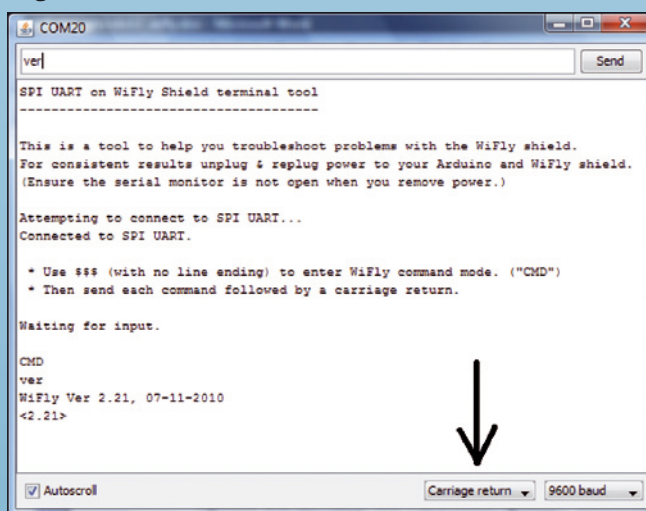
Fig 5 *- Richiesta versione firmware.*

del firmware del modulo; avrete, così, la certezza che tutto funziona correttamente.
La segnalazione dei tre LED dello shield vi permetterà di capire rapidamente lo stato di funzionamento; la **Tabella 1** riepiloga le varie condizioni segnalate.
Per proseguire sarà ora necessario disporre di una rete alla quale associarsi; nel nostro caso abbiamo utilizzato un router Wi-Fi della Dlink, al quale è connesso il PC con cui programmiamo Arduino. La piccola rete domestica è così configurata: indirizzo IP del PC = 192.168.0.199, indirizzo IP del router 192.168.0.1.

Tabella 1

condizione	PIO6 LED rosso	PIO5 LED giallo	PIO4 LED verde
Luce fissa			Connesso con TCP
Lampeggio veloce	Non associato	TX-RX attività	Nessun indirizzo IP
Lampeggio lento			Indirizzo IP OK
Spendo	associato		

Per le prime prove configuriamo il router per una rete aperta senza password. A questo punto inviamo il comando "scan" che rileva le reti Wi-Fi in prossimità: dovremmo rilevare la nostra rete domestica.
Per associarsi alla rete è sufficiente inviare il comando "join dlink"; il canale di comunicazione ed il tipo di crittografia saranno impostati in automatico. Assicuratevi che sia presente la scritta DHCP=ON, ad indicare che l'assegnazione degli indirizzi IP alle periferiche della rete sarà svolta in automatico dal router, mentre la scritta IP=192.168.0.100:80 vi indicherà l'indirizzo IP assegnato alla WiFly. Complimenti, Arduino fa parte della vostra rete domestica!
Potete usare i comandi "show net" e "show connection" per conoscere le impostazioni della rete.
Adesso siete pronti per avviare lo sketch denominato *WiFly_autoconnect_terminal*, che provvederà all'associazione automatica alla vostra rete; le uniche impostazioni da fare riguardano il file *credential.h*, nel quale dovrete specificare il nome della vostra rete (*char ssid[] = "dlink")*. Sullo sketch principale dovete indicare che la vostra rete non ha alcuna password di protezione *(!WiFly.join(ssid))*. Caricate lo sketch, lasciategli qualche secondo e se tutto è a posto, su serial monitor comparirà la scritta "associated!".
Provate ora a configurare il vostro router per l'accesso alla rete con credenziali, come ad esempio una protezione di tipo WEP, WPA1 o WPA2; in questo caso specificate sia l'*ssid* che la *passphrase*. Con il nostro router Dlink, come anche segnalato nei forum, in reti protette da password abbiamo riscontrato alcuni problemi. Potete provare ad usare lo sketch, preparato da noi, denominato *wifly_01.pde*, che permette l'associazione automatica ad una rete con la procedura a linea di comando passo-passo, molto valida per una diagnostica.
Il passo successivo è l'avvio dello sketch *wifly_webserver*, che semplicemente permette di visualizzare tramite un qualsiasi browser il

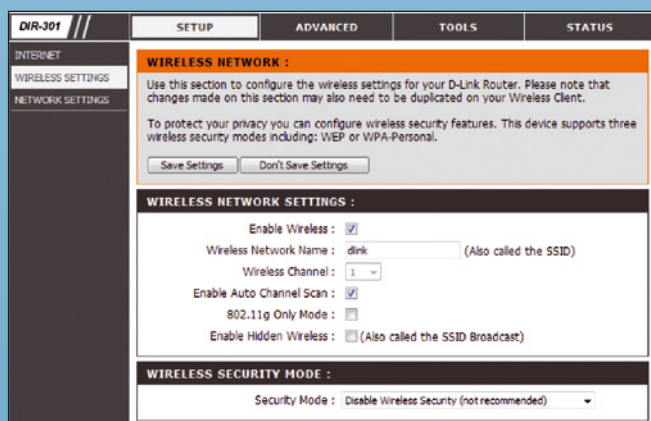



Fig 6 - *Impostazioni del router.*

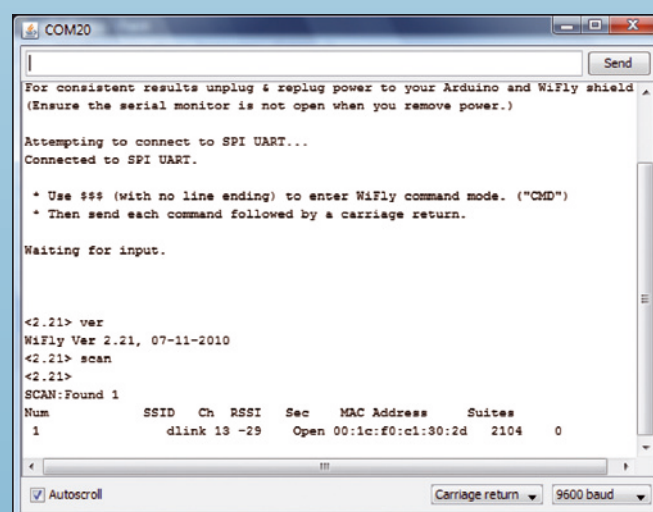



Fig 7 - *Rilevamento reti.*

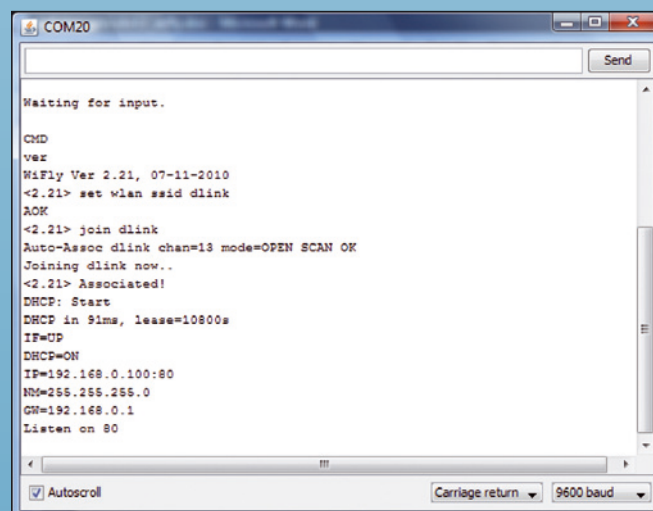



Fig 8 - *Associazione alla rete domestica.*

Fig 9 - *Sketch wifly_01.pde su Serial Monitor.*

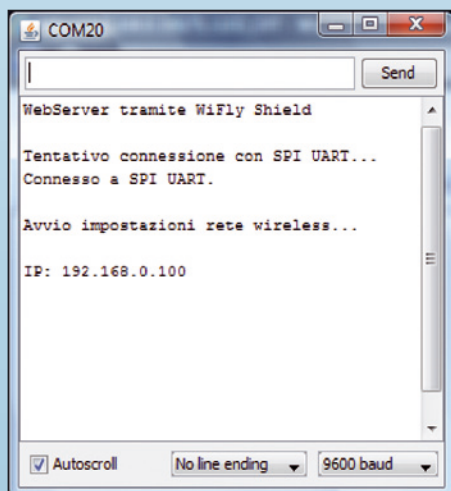



Fig 10 - *Visualizzazione tramite browser relativa allo sketch wifly_01.pde.*

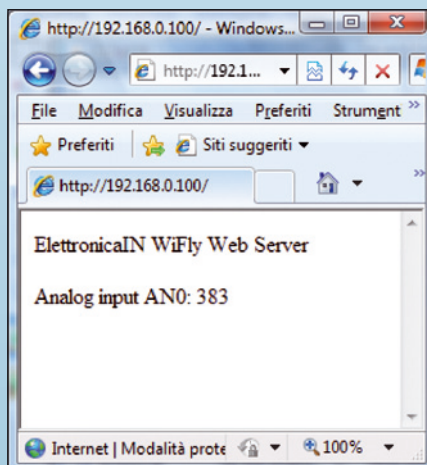



Fig 11 - *Visualizzazione tramite browser relativa allo sketch wifly_02.pde.*

valore degli ingressi analogici di Arduino. Per maggiori dettagli sulle funzionalità di questo e degli altri sketch di esempio, si consiglia di leggere l'articolo del corso su Arduino riguardante l'Ethernet shield. Anche in questo caso abbiamo scritto un nostro sketch, che permette la gestione tramite browser di un LED, connesso all'uscita 2 di Arduino.
Un ulteriore sketch denominato *wifly_04.pde* permette di interagire con una pagina web contenente un menu a tendina, il cui valore selezionato sarà inviato ad Arduino.
Ulteriori esempi sono forniti assieme alla libreria e riguardano le applicazioni di *Web_Client* già descritte in occasione dell puntata del corso dedicata all'Ethernet Shield.
L'utilizzo della WiFly non si limita certamente a quanto qui descritto, ma esistono altri impieghi di un certo interesse; uno di questi è la gestione di una scheda Arduino tramite un telefono cellulare. Dedicheremo a ciò le prossime pagine di questa puntata del corso. In pratica intendiamo gestire ingressi e uscite di Arduino tramite un iPhone, senza però passare da un PC oppure da un router sempre accesi per avere la connessione in rete Wi-Fi. Il nostro approccio, questa volta è diverso: vogliamo che sia la nostra WiFly a creare una rete di tipo ad-hoc tra essa e una periferica esterna, che nel nostro caso è un iPhone, aprendo un canale di comunicazione nello standard UDP.
Per prima cosa è necessario configurare il modulo RN-131G in modo da renderlo capace di creare una rete ad-hoc tra due dispositivi.

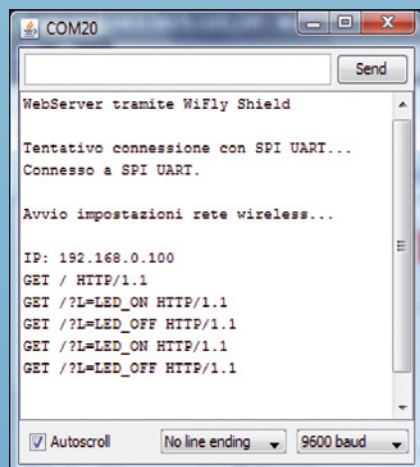



Fig 12 - *Sketch wifly_02.pde su serial monitor.*

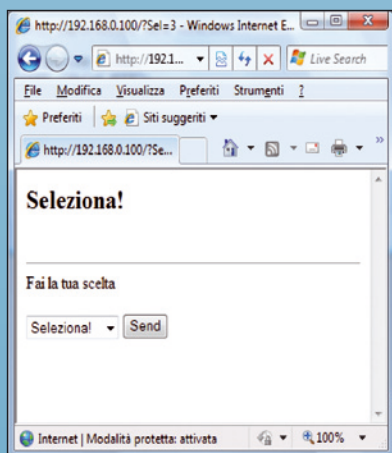



Fig 13 - *Visualizzazione tramite browser relativa allo sketch wifly_04.pde.*

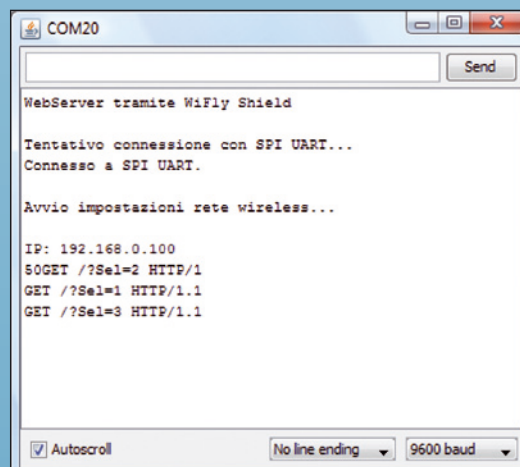
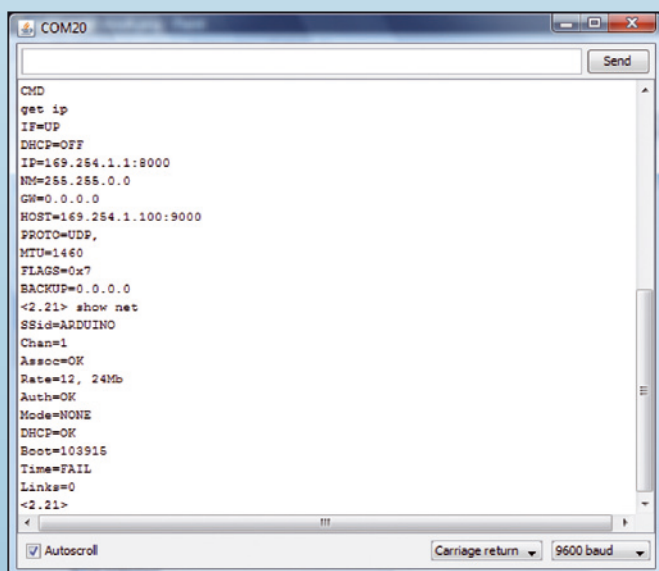



Fig 14 - *Sketch wifly_04.pde su serial monitor.*

Fig 15 *- Configurazione rete adhoc.*

Fig 17 *- Rilevamento rete da parte di iPhone.*

Per fare questo, è necessario utilizzare lo sketch SPIUartTerminal, come già descritto precedentemente, ed inviare la seguente configurazione che andiamo a descrivere passo-passo:

- *Set wlan ssid ARDUINO*; imposta il nome della rete visibile dal dispositivo che vorrà connettersi;
- *Set wlan join 4*; imposta il tipo di rete in modalità ad-hoc;
- *Set wlan chan 1*; imposta il numero di canale utilizzato per la comunicazione Wi-Fi;
- *Set ip adress 169.254.1.1*; imposta l'indirizzo IP del WiFly nella rete (local);
- *Set ip netmask 255.255.0.0*; imposta la maschera di sottorete;
- *Set ip dhcp 0*; disabilita il DHCP;
- *Set ip protocol 1*; imposta il protocollo di comunicazione su UDP;
- *Set ip host 169.254.1.100*; imposta l'indirizzo del dispositivo esterno (host);
- *Set ip remote 9000*; imposta il numero di porta utilizzato dal dispositivo esterno per ricevere i messaggi inviati da WiFly;
- *Set ip local 8000*; imposta il numero di porta utilizzato da WiFly per ricevere i messaggi dal dispositivo esterno;
- *Save*; salva i valori in memoria permanente;
- *Reboot*; riavvia il sistema e rende operativi i nuovi parametri;

È possibile, a questo punto, interrogare il

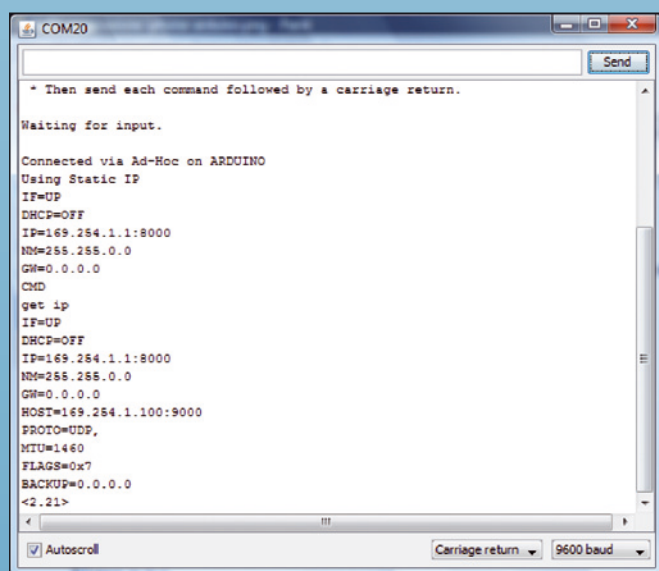



Fig 16 *- Configurazione indirizzi IP rete adhoc.*

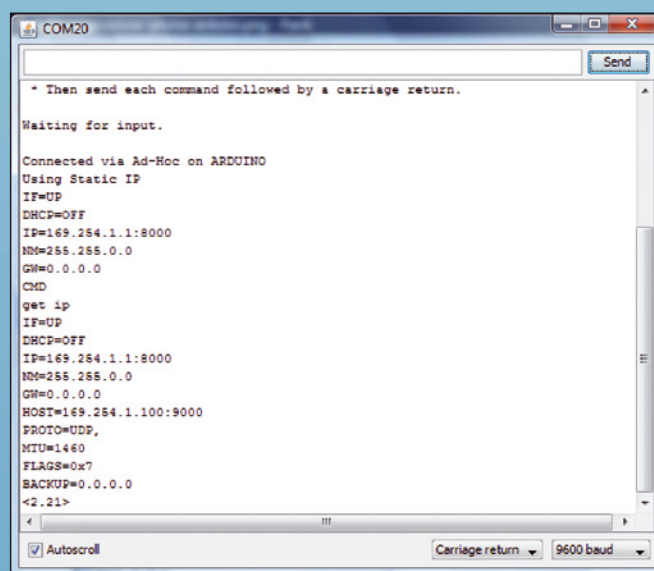
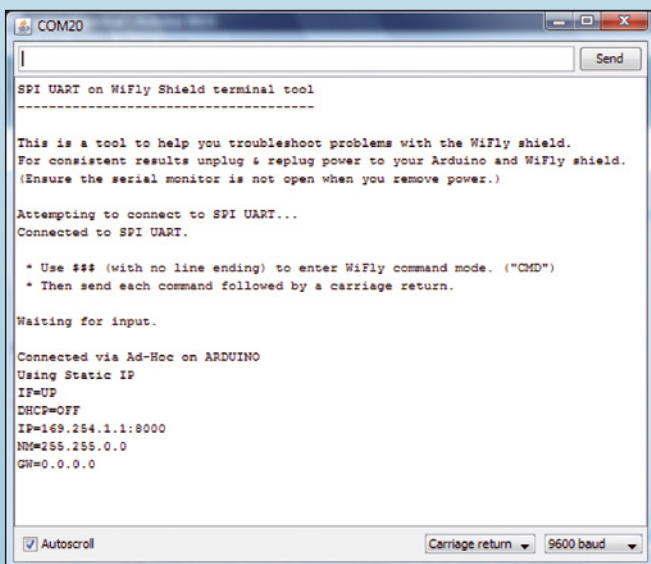



Fig 18 *- Impostazione manuale della rete di iPhone.*

Fig 19 *- Associazione tra WiFly e IPhone.*

WiFly shield, con i comandi *get ip e show net*, per verificare l'effettiva funzionalità dei nuovi parametri (figure 15 e 16).
A questo punto dovete intervenire sull'iPhone per impostare i parametri della nuova rete; accedete alla sezione Wi-Fi del menu impostazioni e in esso impostate la modalità Wi-Fi su ON ed attendete che l'iPhone rilevi la presenza della nuova rete, identificandola con il nome ARDUINO, da noi stessi assegnato nella programmazione del WiFly.
Assegnate manualmente l'indirizzo IP statico a 169.254.1.100 e la maschera di sottorete al valore 255.255.0.0; i campi *DHCP* e *Router* vanno lasciati vuoti. Cliccate sulla connessione per ottenere l'associazione alla nuova rete; un segno di spunta apparirà di fianco al nome della rete.

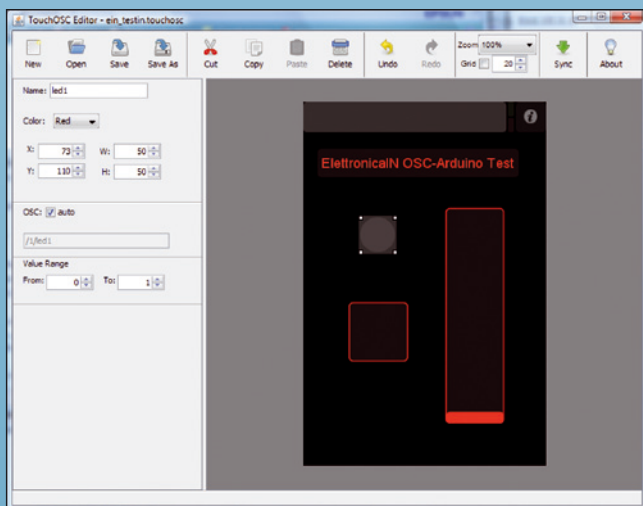



Fig 20 *- Utilizzo di touch editor.*

Fig 21 *- Schermata dell'applicazione per iPhone.*

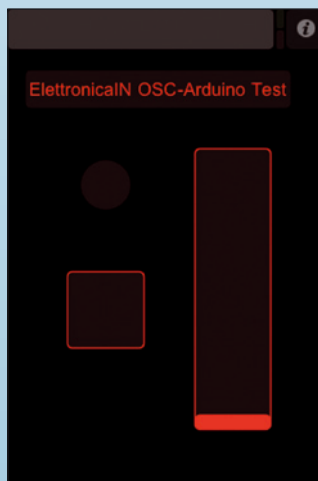



Fig 22 *- Impostazioni per TouchOSC.*

A questo punto dobbiamo occuparci dell'applicazione da far girare su iPhone per poter inviare messaggi tramite la nuova rete creata. Il modo più semplice prevede di utilizzare il protocollo UDP per il trasporto dei dati codificati secondo standard OSC. In APP Store sono disponibili diverse applicazioni più o meno sofisticate che permettono di inviare questo tipo di messaggi, come ad esempio *TouchOSC, IOSC, OSCEmote, MRMR_OSC_controller*.
Per il nostro scopo abbiamo preferito l'applicazione *TouchOSC* che, anche se più costosa, dispone di un ottimo supporto on-line; inoltre è facilmente personalizzabile tramite l'apposito editor (*touchosc-editor*) di immediato e facile

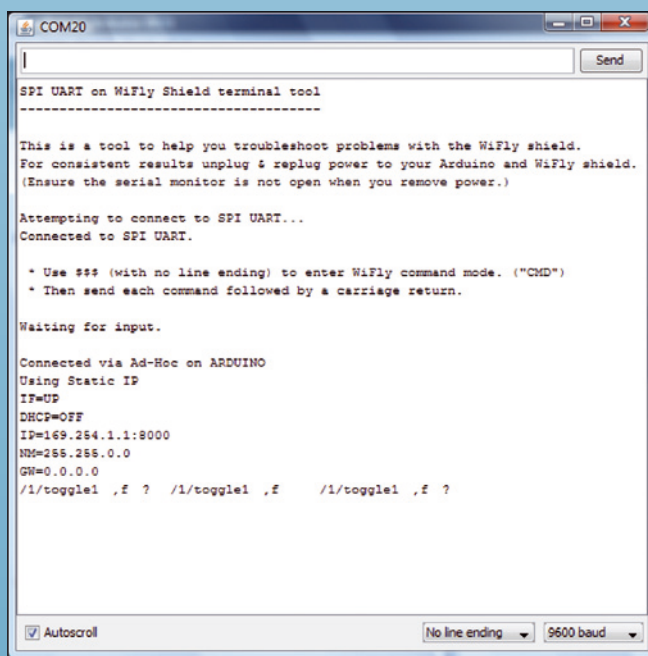
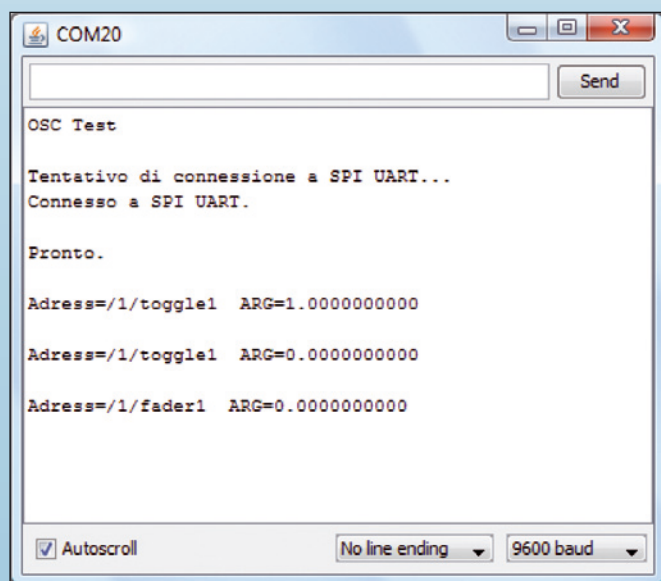



Fig 23 *- Ricezione messaggi OSC da iPhone.*

Fig 24 - *Risultati dello sketch wiflyosc_01.*

utilizzo tramite PC. Il sito di riferimento è *www.hexler.net.*
Tramite l'editor di Touchosc realizzate una schermata con un LED, un pulsante ed un fader; i nomi predefiniti corrispondono agli indirizzi cui dovremo fare riferimento per sviluppare l'applicazione per Arduino:
- per il LED "/1/LED1";
- per il pulsante "/1/toggle1";
- per il fader "/1/fader1".

Caricate sull'iPhone questa nuova schermata seguendo le indicazioni riportare sul sito di riferimento.
Impostate su Touch OSC i parametri di funzionamento come segue: Host=169.254.1.1, Port (outgoing)=8000, Port(incoming)=9000, Local IP address viene assegnato in automatico. Ora siete pronti per far funzionare il tutto. Arduino è operativo con lo sketch *SPIUartTerminal*, WiFly è associata con l'iPhone e *TouchOSC* è avviato e configurato.
Avviate Serial Monitor di Arduino per vedere i messaggi inviati dall'iPhone e in arrivo sulla WiFly. Provate a cliccare sul pulsante (*toggle1*) per inviare un primo messaggio; su Serial Monitor vedrete comparire una stringa contenente l'indirizzo dell'elemento che lo ha generato ed il corrispondente valore numerico.
Come potete vedere, il protocollo OSC, almeno nella sua forma elementare, è molto semplice; purtroppo non esiste ancora una apposita libreria che sia in grado di lavorare con messaggi OSC provenienti dal WiFly shield.
Fortunatamente il protocollo OSC è ben documentato ed il sito di riferimento *http://opensoundcontrol.org* ci permette di comprendere al meglio questo sistema di comunicazione, nato principalmente per far comunicare tra loro dispositivi multimediali e strumenti musicali digitali.
Il messaggio OSC, spedito dall'applicazione *touch-OSC*, è composto da tre campi, il primo dei quali contiene l'indirizzo dell'elemento coinvolto nell'invio del messaggio (pulsante, fader, ecc…). Un secondo campo contiene indicazioni su quali saranno gli argomenti numerici successivi (integer, float, string), mentre il terzo campo contiene il dato, che nel nostro caso è un solo valore numerico di tipo float che riporta lo stato del pulsante (0.0=OFF 1.0=ON) oppure il valore del fader (0.0=cursore in basso 1.0=cursore in alto).
Nel protocollo OSC il messaggio deve essere composto da gruppi di 4 byte, pertanto potrebbe contenere anche dei caratteri NULL che devono essere opportunamente separati dai campi di importanza.
Ad esempio, premendo il pulsante toggle1 il messaggio inviato sarà:

2f (/)	31 (1)	2f (/)	74 (t)
6f (o)	67 (g)	67 (g)	6c (l)
65 (e)	0 ()	0 ()	0 ()
2c (,)	66 (f)	0 ()	0 ()
0 ()	3f (?)	0 ()	0 ()

La prima stringa ricevuta "/1/toggle1" indica che è stato premuto il pulsante, segue il campo relativo all'argomento ",f" ad indicare che segue un unico dato di tipo float a 32 bit. I seguenti caratteri essendo appunto un valore float a 32 bit non sono identificabili come caratteri ma vanno invece analizzati come un insieme di bit.
Il passo successivo consiste nel creare uno sketch che estrapoli, dal messaggio OSC, l'indirizzo ed il valore numerico; lo sketch si chiama *wiflyosc_01* ed è disponibile assieme ai file di questo articolo.
Nello sketch contenuto nel **Listato 1**, viene usata la libreria *wifly.h* solo per la gestione della comunicazione con la WiFly; il resto del listato riguarda l'elaborazione dei caratteri provenienti dalla comunicazione SPI.
Il loop principale provvede alla ricezione dei caratteri in arrivo per poi richiamare la funzione oscRXHandler che ha il compito

Listato 1

```
void loop() {
  byte incomingByte;
  // Aspetto la ricezione dei dati dalla wifly:
  while (SpiSerial.available() > 0) {
    incomingByte = SpiSerial.read() & 0xFF; // leggo un byte
    oscRxHandler(incomingByte); // Elaborazione del carattere
  }
}
```

di interpretarli. Alla funzione *oscRXHandler* è associata una variabile di stato `OSC_RXOP`, la quale tiene traccia dello stato di ricezione, a seconda che stia arrivando l'indirizzo, il tipo di dato oppure il valore, contenuti nel messaggio OSC.
L'arrivo del quarto byte del dato identifica la fine del messaggio OSC e le variabili `oscRxData` e `oscRxFloatArg` contengono, rispettivamente, il valore numerico del dato ed il suo indirizzo.
Adesso completiamo lo sketch aggiungendo la possibilità di attivare un LED, connesso all'uscita 2 di Arduino, tramite il pulsante *toggle1* di TouchOSC. Aggiungiamo anche la possibilità di attivare da Arduino il *Led1* di TouchOSC e sempre da Arduino andare a modifica la scritta su *Label1*.
Lo sketch si chiama *wiflyosc_02* e attraverso Serial monitor potete inviare i seguenti comandi:
- carattere 'a' =attiva LED1 su touchOSC;
- carattere 'b' = spegne LED1 su touchOSC;
- carattere 's' =scrive su Label1 di touchOSC la scritta "ciao";
- carattere 'd' = scrive su Label1 di touchOSC la scritta "mondo".

In questo sketch sono presenti le procedure `oscSendFloat` e `oscSendString` che provvedono all'invio di messaggi OSC verso iPhone.

Ad esempio, per accendere il *led1* su touchOSC è sufficiente scrivere:
`oscSendFloat` ("/1/led1", 1.0)
mentre la riga di codice `oscSendString` ("/1/label1", "ciao") consente di modificare la scritta attuale su *label1* di *touchOSC* con la scritta "ciao".
Per la ricezione dei messaggi è stata predisposta la procedura `oscReceiveFloat(char * msg, float value)` che viene richiamata ogni qualvolta arriva un messaggio da *touchOSC*.
Nel nostro caso è utilizzata per attivare, tramite il pulsante *toggle1* di touchOSC, il LED collegato al contatto 2 di Arduino (**Listato 2**).
Facilmente, potrete adattare questi sketch per altri generi di applicazione. Ad esempio, oltre al controllo in remoto di svariati tipi di robot, l'interesse potrebbe spostarsi sulla domotica, in quanto la WiFly è configurata per associarsi ad un iPhone con un specifico indirizzo IP non appena si trova nel suo raggio d'azione.
Immaginate di essere appena rincasati dal lavoro e di trovarvi, col vostro cellulare, a portata di WiFly: subito il vostro iPhone diventa automaticamente il controller di casa vostra, permettendovi dall'apertura del cancello alla gestione delle tende da sole, fino al comando dell'impianto di riscaldamento. Il tutto nelle vostre mani. ■

Listato 2

```
void oscReceiveFloat(char * msg, float value)
{
char adress1[] = "/1/toggle1";
  if(strncmp(msg,adress1,strlen(adress1))==0) {
    if (value==1.0) {
      Serial.println("P1=ON");
      digitalWrite(2, HIGH);   // accende LED
      }
    if (value==0.0) {
      Serial.println("P1=OFF");
      digitalWrite(2, LOW);   // spegne LED
    }
  }
  return;
}
```